Free Pascal (autore: Vittorio Albertoni)

Premessa

Il linguaggio Pascal fu creato nel 1970 (due anni prima che Dennis Ritchie creasse il linguaggio C), da Niklaus Wirth e Kathleen Jénsen a scopo didattico: Wirth era infatti un insegnante di informatica e creò il Pascal per insegnare le basi della programmazione strutturata.

Il nome rende omaggio al filosofo Blaise Pascal, che, tra le tante cose che pensò e scrisse, inventò anche la prima calcolatrice.

La creatura servì per qualche cosa che andava ben oltre la didattica: gran parte dei sistemi operativi per il Macintosh e per Microsoft Windows sono infatti stati scritti in Pascal.

All'epoca esisteva già il linguaggio BASIC, concepito per «beginners», cioè per principianti, ma Wirth non lo riteneva adatto all'insegnamento in quanto era sì facile da imparare ma non aveva strutture dati avanzate e non incoraggiava abbastanza ad analizzare il problema prima di scrivere effettivamente il codice. Non era cioè considerato un linguaggio da programmazione strutturata.

Con l'avvento del PC IBM dotato di hard disk interno, nel 1983, la Società di informatica Borland mette in commercio un compilatore per il linguaggio Pascal, denominato Turbo Pascal, alludendo alla velocità di compilazione.

Nel 1995 Turbo Pascal confluisce nell'ambiente di sviluppo Delphi, sempre in casa Borland.

Nel 1997, Florian Paul Klämpfl, implementando il Pascal della Borland, crea una versione libera del compilatore, denominata FPK Pascal, utilizzando le iniziali del proprio nome.
Vista la natura libera del progetto e il fatto che immediatamente si dedicano al progetto stesso altri collaboratori, il nome del compilatore viene subito convertito in FPC, che sta per Free Pascal Compiler e il linguaggio, pur debitore in toto al linguaggio Turbo Pascal, si chiama Free Pascal.

Nel 1999, Cliff Baeseman, Shane Miller e Michael A. Hess creano il progetto Lazarus, risuscitando (da qui il nome) un fallito progetto di clonazione dell'ambiente di sviluppo Delphi, dotato di un ambiente visuale di progettazione di interfaccia grafica utente.

Grazie a tutto ciò, il mondo del software libero vanta un ambiente di sviluppo di tutto rispetto, multi-piattaforma, in quanto disponibile per tutti i sistemi operativi, che fa dire ai suoi sostenitori «Write once compile everywhere» (Scrivi una volta e compila ovunque) in contrapposizione al motto «Write once run everywhere» (Scrivi una volta e esegui ovunque) dei sostenitori di Java.

La differenza sta nel fatto che i programmi scritti in Java girano ovunque, purché su ciascun computer sia installata la macchina virtuale Java, mentre un programma Pascal/Lazarus compilato su un computer dotato di un certo sistema operativo gira ovunque sia installato quel sistema operativo, anche se sul computer non sono installati né il compilatore Pascal né Lazarus.

In questo manualetto presento quanto serve a un dilettante per capire come funziona il linguaggio Free Pascal e per costruire qualche semplice programma.

A chi si appassioni e voglia approfondire suggerisco l'indirizzo *http://www.lazaruspascal.it/*, che è il sito della comunità italiana e vi si trova parecchia documentazione nella nostra lingua.

Il massimo della documentazione, in lingua inglese, si trova su *https://www.freepascal.org/*.

Indice

1 Installazione

Il materiale per creare un ambiente di programmazione Free Pascal lo possiamo trovare in due luoghi.

All'indirizzo *https://www.freepascal.org/*, nella sezione DOWNLOAD, è disponibile il compilatore.

Vi troviamo gli installer, suddivisi per CPU e sistema operativo, dell'ultima versione disponibile nel momento in cui scrivo (novembre 2020), la 3.2.0.

Con questa installazione abbiamo disponibile il compilatore e un IDE (ambiente di sviluppo integrato) un tantino antiquato e che ci consente di creare programmi per console, pur abbelliti con una grafica rudimentale. Non abbiamo a disposizione strumenti per creare interfacce utente grafiche (GUI) vere e proprie.

Chi usa Linux, se si accontenta di questi strumenti, può benissimo installare la versione disponibile nell'installatore di software del sistema operativo, denominata `fpc`.

Se prevediamo di dover creare anche programmi dotati di interfaccia utente grafica ignoriamo quanto detto e andiamo all'indirizzo *https://www.lazarus-ide.org/*, dove, nella sezione DOWNLOADS, troviamo quanto serve per disporre dell'ambiente di sviluppo integrato Lazarus. L'ultima versione disponibile è la 2.0.10 e si appoggia al compilatore 3.2.0.

L'installer per il sistema operativo Windows consta di un unico eseguibile lanciando il quale installiamo tutto ciò che serve.

Per i sistemi operativi Linux e Mac OS X occorre scaricare e lanciare più installer (per il compilatore, per Lazarus e per la documentazione necessaria alla code completion).

Anche Lazarus si trova nei repositories delle varie versioni del sistema operativo Linux e lo si potrebbe installare con l'installatore di software. Il ricorso ai file suggeriti dal sito di Lazarus ci offre tuttavia migliori garanzie di fare bene con le dipendenze e di avere un ambiente perfettamente funzionante.

2 Come funziona

I programmi che produciamo con il linguaggio Free Pascal sono eseguibili, cioè funzionano senza alcun interprete (che è invece necessario per programmi in Java, JavaScript, Python, ecc.) ed anche se sul computer non è installato il compilatore, purché siano stati compilati su un sistema operativo dello stesso tipo di quello del computer su cui lavoriamo: un programma compilato su Windows non gira su Linux o su Mac e viceversa.

La compilazione è la traduzione in linguaggio macchina del programma che abbiamo scritto in Free Pascal, un linguaggio che ha sintassi e regole particolari, che vedremo, ma è di tipo umano, in lingua inglese.

Una volta installato il compilatore, da solo o integrato nell'ambiente di sviluppo Lazarus, il modo più semplice e ruspante di produrre un eseguibile è il seguente:

. si scrive il programma con un editor di testo,

. si salva il file con estensione `.pas`,

. si apre il terminale (così chiamato nei sistemi Linux e Mac OS X e chiamato prompt dei comandi in Windows),

. ci si posiziona nella directory dove è stato salvato il file con estensione `.pas`,

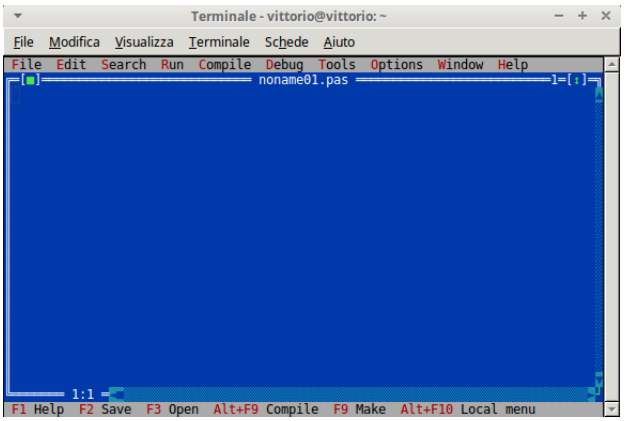
. si scrive il comando `fpc` seguito dal nome del file con estensione `.pas`.

Nella directory dove avevamo salvato il file `.pas` ora troviamo altri due file, uno con lo stesso nome del file `.pas` ma con estensione `.o` e un altro con lo stesso nome del file `.pas` ma senza estensione o, nel sistema operativo Windows, con estensione `.exe`.

Il file con estensione `.o` è il file oggetto, file intermedio per effettuare la compilazione eventualmente linkando altri file oggetto, e, al nostro livello dilettantesco, non ci interessa.

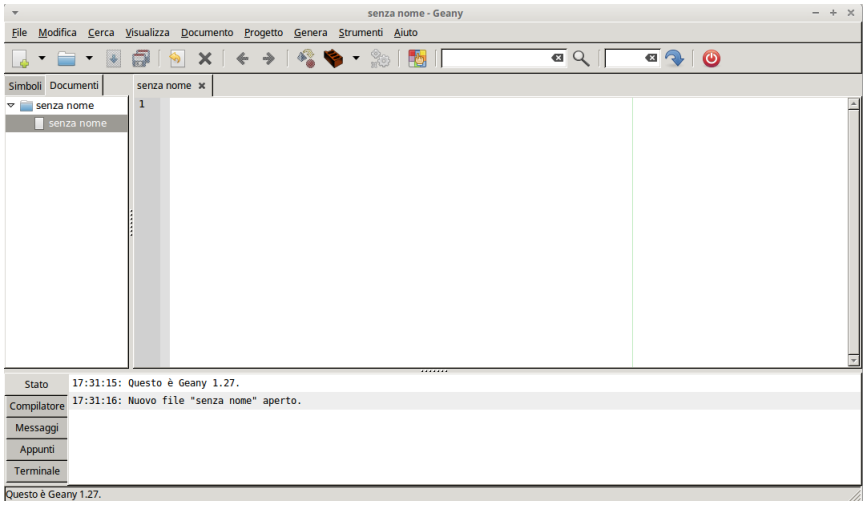
Il file senza estensione o con estensione `.exe` è l'eseguibile ed è il file che, trasferito per semplice copiatura su un altro computer con lo stesso sistema operativo del computer dove è stato compilato, ci consente di far funzionare il nostro programma su quest'altro computer[1].

Anziché usare un qualsiasi editor di testo per scrivere il programma per poi compilarlo richiamando il compilatore possiamo utilizzare lo strumento integrato che ci offre Free Pascal. Si tratta di un programma che parte con il comando `fp`.
La finestra di lavoro è la seguente



Come si vede, oltre alla zona di editing, quella centrale colorata in azzurro, nella quale siamo aiutati da un pochino di code completion, abbiamo dei menu per memorizzare il file (FILE), per compilarlo (COMPILE) e per eseguirlo (RUN). Per quando saremo programmatori più esperti anche per lo scorrimento del programma alla ricerca di errori (DEBUG).

Personalmente preferisco programmare in Free Pascal con l'editor Geany, più moderno rispetto a quello che passa il convento e con una code completion più ricca, pur se ispirata a Turbo Pascal.
L'area di lavoro si presenta così



Passando il mouse sulle icone della barra degli strumenti sotto la barra dei menu ci viene spiegato a cosa serve ciascun strumento.
Anche in questo caso possiamo scrivere il programma, compilarlo e provarlo senza muoverci dall'editor.

[1] La semplice copiatura non basterà per far funzionare il programma in quanto si dovranno superare le eventuali barriere di sicurezza (un file eseguibile può essere un virus). Nel sistema Windows occorrerà seguire le istruzioni per assicurare il sistema che ciò che si sta facendo è regolare. Nei sistemi Linux e Mac OS X, eredi della formidabile sicurezza Unix, occorrerà confermare che si tratta di un file eseguibile nei modi previsti (per esempio, posizionati nella directory dove è stato copiato il file, scrivendo a terminale il comando `chmod 755` seguito dal nome del file).

Per poter disporre della code completion dobbiamo salvare il file prima di cominciare a scrivere dandogli estensione `.pas`.

Con questi strumenti, conoscendo le basi del linguaggio Free Pascal, possiamo agevolmente creare programmi console, cioè programmi che interagiscono con l'utente attraverso il terminale, quello che in Windows si chiama Prompt dei comandi.

Con il piccolo sforzo di conoscere una facile estensione del linguaggio base possiamo abbellire graficamente il terminale su cui lavorare.

Per creare programmi dotati di interfaccia grafica dobbiamo ricorrere a Lazarus.

Lazarus contiene un ambiente visuale di costruzione dell'interfaccia grafica che ci consente di disegnare l'interfaccia stessa con le sue componenti (pulsanti, finestrelle, ecc.) occupandosi lui di tradurre il disegno in istruzioni per il compilatore. A noi rimane da scrivere, utilizzando le basi del linguaggio, le procedure necessarie per eseguire quanto sottende l'interfaccia stessa, per esempio ciò che deve succedere quando si clicca su un pulsante.

Con Lazarus possiamo ovviamente creare anche semplici programmi da console, ma direi che non vale la pena scomodare Lazarus per così poco e per queste cose semplici facciamo sicuramente meglio con gli strumenti visti prima.

3 Basi del linguaggio

Il compilatore di Free Pascal non distingue tra lettere maiuscole e lettere minuscole, per cui possiamo scrivere le istruzioni in maiuscolo, in minuscolo, con la sola iniziale maiuscola (stile tipico ma non tassativo del linguaggio Pascal classico), a nostro piacimento. La mia personale preferenza è per le lettere minuscole e, nel seguito, mi atterrò a questa preferenza.

Se vogliamo inserire commenti nel codice (frasi e dizioni, che il compilatore non veda, solo per commentare il codice) dobbiamo racchiuderli tra parentesi graffe (`{` e `}`) oppure, se contenuti su una sola linea, anteponendovi il simbolo `//`.

3.1 Tipi di dato

Free Pascal, buon erede del Pascal standard, è probabilmente il linguaggio di programmazione che contempla la maggiore varietà di tipi di dato e addirittura prevede che il programmatore ne possa costruire di propri.

Per il dilettante alle prime armi mi limito a citare quelli più ricorrenti e con i quali già si può fare molto.

3.1.1 Numeri

Dei tanti tipi numerici previsti, retaggio dei tempi in cui l'hardware disponibile imponeva molta attenzione per evitare spreco di memoria, mi limito a proporne una selezione che ritengo adeguata al principiante e all'hardware moderno[2].

Numeri interi Per maneggiare numeri interi di dimensione ridotta, di valore inferiore a 65.535, abbiamo a disposizione il tipo **integer**. Una variabile di questo tipo occupa 2 bytes di memoria.
Per maneggiare numeri interi di valore compreso tra -2.147.483.648 e 2.147.483.647 abbiamo a disposizione il tipo **longint**.
Se rinunciamo ad utilizzare il segno possiamo optare per il tipo **longword** ed arrivare a maneggiare numeri di valore compreso tra 0 e 4.294.967.295. Una variabile di questi tipi occupa 4 bytes.

[2]Per giustificare la maniacale attenzione dei primi linguaggi di programmazione nei confronti dell'occupazione di memoria rammento che solo una trentina di anni fa, il mio primo personal computer, un glorioso Philips TC 100, aveva un disco fisso di 24 MB, destinati ad ospitare sistema operativo, applicativi e quant'altro. Oggi non vi si potrebbe caricare nemmeno il file audio di una canzonetta di tre minuti.

Per i grandi numeri con segno abbiamo a disposizione
. il tipo **int64** (numeri tra -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807),
. rinunciando al segno, il tipo **qword** (numeri tra 0 e 18.446.744.073.709.551.615).
Una variabile di questi tipi occupa 8 bytes.

Numeri reali Quando abbiamo a che fare con numeri reali, detti anche a virgola mobile o decimali, ciò che importa maggiormente è la precisione, cioè il numero di cifre significative su cui contare, cifre comprensive di quelle prima e dopo la virgola di separazione (che nel linguaggio Free Pascal è un punto).
Se ci bastano 8 cifre significative abbiamo a disposizione il tipo **single** e la variabile occupa 4 bytes.
Se vogliamo 16 cifre significative abbiamo a disposizione il tipo **double** e la variabile occupa 8 bytes.
Possiamo arrivare a 20 cifre significative con il tipo **extended** e la variabile occupa 10 bytes.

Attraverso i tipi qui richiamati, in notazione scientifica, è possibile rappresentare numeri composti da tantissime cifre ma ricordiamo, in ogni caso, che le cifre significative sono quelle prima indicate.
Per esempio, se calcoliamo il fattoriale del numero 32, che è costituito da 36 cifre, utilizzando il tipo intero qword otteniamo un risultato completamente sballato in quanto fuori dal limite rappresentabile con quel tipo (18.446.744.073.709.551.615). Se lo calcoliamo utilizzando il tipo extended otteniamo un risultato che è esatto solo per le prime 20 cifre.

3.1.2 Caratteri

Il tipo **char** può assumere uno dei 256 valori (da 0 a 256) dell'insieme di caratteri ASCII. Tali valori possono essere passati indicando il carattere tra apici semplici (`'`), indicando il codice decimale del carattere preceduto dal simbolo `#` oppure indicando il codice esadecimale del carattere preceduto dal simbolo `$`.
Il carattere C può essere passato con `'C'`, `#67` oppure `$43`.

3.1.3 Stringhe

Il tipo **string** definisce una sequenza di caratteri racchiusa tra apici semplici (`'`).
La sua dimensione massima è di 255 caratteri ma, per risparmiare memoria, si può indicare una dimensione massima inferiore, facendo seguire al nome del tipo, tra parentesi quadre, questa dimensione.
`string` identifica una stringa di un massimo di 255 caratteri;
`string[12]` identifica una stringa di un massimo di 12 caratteri;
`string[300]` genera un errore di compilazione.

Per inserire un apice semplice in una stringa occorre scriverlo due volte (la stringa 'D''Angelo' contiene il nome D'Angelo).

3.1.4 Boolean

Il tipo **boolean** può assumere solo due valori: `true` e `false`.

* * *

I tipi visti sopra, ad eccezione dei tipi int64, qword e string, sono i così detti tipi semplici predefiniti del Pascal standard.

I tipi int64 e qword sono stati introdotti da Free Pascal.

Il tipo string è stato introdotto dal Turbo Pascal e da qui è passato a Free Pascal. In realtà è un vettore (array) di elementi di tipo char e, nel Pascal standard veniva gestito così, con molta complicazione. Proprio per evitare queste complicazioni non cito qui i tipi composti, come array, enumerazioni, set, ecc. previsti dal Free Pascal.

3.2 Costanti e Variabili

Le costanti e le variabili sono zone di memoria riservate a contenere i dati oggetto delle elaborazioni previste dal programma.

3.2.1 Costanti

I valori assegnati alle costanti non sono modificabili nel corso del programma.

Una costante si definisce con la seguente sintassi:
`const` <identificatore> = <valore>;
dove
<identificatore> è il nome che assegniamo alla costante e serve per richiamarla nel corso del programma,
<valore> è esprimibile con un qualsiasi valore di tipo numerico o di tipo stringa.
Se il valore numerico è superiore al massimo di quello gestibile, solo le sue prime venti cifre vengono registrate esattamente.
Se il valore stringa è superiore a 255 caratteri, solo i primi 255 caratteri vengono registrati.

3.2.2 Variabili

La variabile è destinata a contenere valori che possono essere modificati nel corso del programma.

Una variabile si definisce con la seguente sintassi:
`var` <identificatore>: <tipo>;
dove
<identificatore> è il nome che assegniamo alla variabile e serve per richiamarla nel corso del programma,
<tipo> è il tipo di dato che la variabile è destinata a contenere.

Possiamo definire ad un tempo variabili dello stesso tipo con la sintassi:
`var` <identificatore>, <identificatore>, <identificatore>, ...: <tipo>;.

Infine possiamo inizializzare una variabile con la sintassi:
`var` <identificatore>: <tipo> = <valore>;,
dove
<valore> deve rispettare il tipo indicato e i suoi limiti.

Bisogna fare molta attenzione alla scelta del tipo con riguardo ai limiti dei dati con esso gestibili.

Infatti, se, indicato un tipo, al momento dell'inizializzazione o nel corso del programma assegniamo alla variabile un valore di un tipo diverso (ad esempio un numero decimale anziché un intero), viene segnalato errore di runtime e il programma si ferma.
Se, invece, indicato un tipo, al momento dell'inizializzazione o nel corso del programma assegniamo alla variabile un valore superiore a quello gestibile dal tipo indicato, avremo risultati completamente sballati senza che lo sappiamo.

Da qui il consiglio di qualcuno: se non abbiamo contezza di quanto grande possa essere un valore da inserire in una variabile, dichiariamo questa variabile di tipo qword anche in presenza di un numero intero, in modo da potervi registrare esattamente almeno le sue prime 20 cifre significative.

3.3 Operatori

L'operatore è un simbolo che, inserito tra valori, indicati letteralmente o rappresentati da costanti o da variabili, produce un risultato.

3.3.1 Operatori aritmetici

Si utilizzano tra tipi numerici. I più comuni, elencati in ordine di precedenza, sono:
`*` per la moltiplicazione,
`/` per la divisione, con l'operando destro diverso da zero,
`div` per il quoziente intero tra due numeri interi, con quello di destra diverso da zero,
`mod` per il resto della divisione tra due numeri interi, con quello di destra diverso da zero,
`+` per l'addizione,
`-` per la sottrazione.

L'operatore `+` tra due stringhe le concatena.

3.3.2 Operatori relazionali

Si utilizzano per confrontare tra loro grandezze dello stesso tipo e il risultato del confronto è un valore booleano: `true` se la relazione è vera, `false` se la relazione non è vera.
`=` uguale a,
`>` maggiore di,
`<` minore di,
`>=` maggiore o uguale a,
`<=` minore o uguale a,
`<>` diverso da.

3.3.3 Operatori logici

Si utilizzano per operare su variabili di tipo booleano e restituiscono un valore booleano.
`and` fornisce il risultato `true` se entrambi gli operandi sono `true`,
`or` fornisce il risultato `false` se entrambi gli operandi sono `false`,
`xor` fornisce il risultato `true` solo se un operando è `true` e l'altro è `false`.

3.4 Procedure e funzioni predefinite

La procedura è un comando per svolgere determinate azioni e la funzione è un comando al quale si passano dei valori (detti parametri o argomenti) per ottenere un risultato.

Il linguaggio Free Pascal contiene numerose funzioni predefinite. Le espongo classificandole per tipi di dato.

3.4.1 Funzioni aritmetiche

Consentono la manipolazione di dati numerici.
`abs(n)` restituisce il valore assoluto del valore numerico n,
`exp(n)` restituisce la potenza ennesima del numero e,
`frac(n)` restituisce, sotto forma di numero reale, la parte decimale di n,
`int(n)` restituisce, sotto forma di numero reale, la parte intera di n,
`round(n)` restituisce, sotto forma di numero intero, la parte intera di n arrotondata,
`trunc(n)` restituisce, sotto forma di numero intero, la parte intera di n senza arrotondamenti,
`ln(n)` restituisce il logaritmo naturale di n,
`pi` restituisce il valore della costante π,
`sqr(n)` restituisce il quadrato di n,
`sqrt(n)` restituisce la radice quadrata di n.
Per la potenza ennesima di x possiamo usare la combinata `exp(ln(x)*n)`.
Per la radice ennesima di x possiamo usare la combinata `exp(ln(x)/n)`.

3.4.2 Funzioni trigonometriche

Esprimendo n in radianti abbiamo le funzioni dirette `sin(n)` e `cos(n)`.
Abbiamo poi `arctan(n)` che restituisce, in radianti, il valore dell'angolo avente per tangente n.
Per la tangente possiamo usare la combinata `sin(n)/cos(n)`.
Per l'arcoseno possiamo usare la combinata `arctan(n/sqrt(1-sqr(n)))` e per l'arcocoseno la combinata `arctan(sqrt(1-sqr(n))/n)`.

3.4.3 Funzioni per manipolare stringhe

Forniscono informazioni o esplicitano espressioni relative alle stringhe.
`concat(<stringa>, <stringa>,...)` concatena stringhe (può convenientemente essere sostituita dall'operatore +),
`copy(<stringa>, <posizione>, <quanti>)` copia da <stringa>, a partire da <posizione>, <quanti> caratteri (la posizione va determinata rammentando che i caratteri della stringa sono collocati a partire dalla posizione 0),
`delete(<stringa>, <posizione>, <quanti>)` cancella da <stringa>, a partire da <posizione>, <quanti> caratteri (la posizione va determinata rammentando che i caratteri della stringa sono collocati a partire dalla posizione 0),
`lentgh(<stringa>)` restituisce un intero corrispondente al numero dei caratteri della stringa,
`pos(<cosa>, <stringa>)` restituisce il valore posizionale del primo carattere di una sottostringa <cosa>.

3.4.4 Generazione di numeri casuali

Per generale un numero casuale compreso in un intervallo occorre inizializzare il generatore con il comando
`randomize`, senza parametri (infatti siamo in presenza di una procedura e non di una funzione),
e poi generare il numero casuale compreso tra 0 e n con la funzione
`random(n+1)`.

3.5 Istruzioni

Le istruzioni costituiscono la parte principale dello sviluppo del programma.

Se si tratta di istruzioni semplici, occupano una sola riga.

Se si tratta di istruzioni composte, da eseguirsi in blocco, esse si elencano tra la parola chiave `begin` e la parola chiave `end`.

L'istruzione deve tassativamente terminare con il punto e virgola (;).

3.5.1 Istruzioni di assegnazione

Si tratta di istruzioni semplici con le quali si assegnano valori alle variabili.

La sintassi è
`<nome_variabile> := <valore>;`
dove
<nome_variabile> richiama il nome di una variabile precedentemente definita,
<valore> può essere indicato in termini letterali o attraverso una espressione, in ogni caso rispettando rigorosamente il tipo con cui era stata definita la variabile.

Avendo precedentemente definito la variabile `nome` di tipo `string`, l'istruzione `nome := 'Vittorio';` le assegna il valore 'Vittorio'.

Avendo precedentemente definito la variabile `x` di tipo `double`, l'istruzione `x := 7.5;` le assegna il valore 7.5.

Con l'istruzione `x := 3 * 4.5;` le assegneremmo il valore 13,5.

Con l'istruzione x := 2.1 * `exp(ln(27)/3);` le assegneremmo il valore 6,3.

3.5.2 Istruzioni condizionali

Si utilizzano per effettuare la scelta tra possibili azioni, a seconda del verificarsi o meno di particolari condizioni.

if

La sintassi è la seguente
`if (`<espressione_logica>`) then` <istruzione>;
dove
<espressione_logica> è la condizione; la sua collocazione tra parentesi tonde non è d'obbligo nel caso di espressione logica semplice (come a < b) ma lo diventa nel caso di espressioni logiche complesse (come (a<b) and (c>d));
<istruzione> è ciò che deve essere eseguito se la condizione è vera; nel caso di più istruzioni da eseguirsi in blocco vanno elencate tra `begin` e `end`.
Se la condizione non è vera l'istruzione non viene eseguita e si procede con il resto del programma.

Per il caso che la condizione non sia vera possiamo prevedere l'esecuzione di una istruzione alternativa con la sintassi
`if (`<espressione_logica>`) then` <istruzione> `else` <istruzione>;
Notare che il punto e virgola di fine istruzione va messo solo dopo la seconda <istruzione>. Nel caso di blocchi di istruzioni poste tra `begin` e `end`, il punto e virgola va messo solo dopo l'`end` che termina la seconda <istruzione>.

case

Si utilizza per scegliere un'azione da eseguire in corrispondenza del valore che assume un'espressione.

Rispetto all'istruzione precedente, che prevede solo due possibilità per la scelta (espressione vera o espressione falsa), qui possiamo avere un lungo elenco di valori per orientare le scelte.

La sintassi è
`case` <variabile> `of`
<valore>: <istruzione>;
<valore>: <istruzione>;
<valore>: <istruzione>;
`.......`
`end;`

Prima dell'`end` di chiusura dell'istruzione possiamo inserire
`else` <istruzione>;
per avere un default nel caso non sia riscontrato alcun <valore> valido.

In questo esempio abbiamo due variabili stringa: mese, destinata ad ospitare il nome italiano di un mese e traduzione, destinata a ospitare la relativa traduzione in inglese:

```
case mese of
   'Gennaio':  traduzione := 'January';
   'Febbraio':  traduzione := 'February';
   'Marzo':  traduzione := 'March';
   'Aprile':  traduzione := 'April';
   'Maggio':  traduzione := 'May';
   'Giugno':  traduzione := 'June';
   'Luglio':  traduzione := 'July';
   'Agosto':  traduzione := 'August';
   'Settembre':  traduzione := 'September';
   'Ottobre':  traduzione := 'October';
   'Novembre':  traduzione := 'November';
```

```
   'Dicembre':  traduzione := 'December';
else traduzione:= 'non ho capito';
end;
```
Se nella variabile mese inseriamo il valore 'Agosto', nella variabile traduzione viene inserito il valore 'August'. Se inseriamo il valore 'agosto' con la lettera minuscola o 'aosto', nella variabile traduzione viene inserito il valore 'non ho capito', ecc.

3.5.3 Istruzioni di controllo del flusso

Si utilizzano per eseguire più volte, in maniera controllata, un'istruzione o un blocco di istruzioni.

for

Serve quando il ciclo ripetitivo deve essere eseguito un numero predeterminato di volte. La sintassi è:
`for` <contatore>`:=`<partenza> `to` <arrivo> `do` <istruzione>;
dove
<contatore> è una variabile destinata a contenere piccoli valori interi, per brevità è in genere battezzata `i`,
<partenza> è il valore da cui parte la conta,
<arrivo> è il valore in corrispondenza al quale finisce la conta,
<istruzione> è ciò che viene fatto per ogni conta.

Per eseguire 5 volte un'istruzione scriviamo
```
for i := 1 to 5 do <istruzione>;
```

repeat

Serve per eseguire il ciclo ripetitivo fino a quando si verifica un certo evento. La sintassi è:
`repeat` <istruzione> `until` <condizione>;

Anticipando che l'istruzione `read(n)` serve per leggere un numero digitato sulla tastiera e inserirlo nella variabile `n`, con quanto segue sommiamo i numeri via via immessi dalla tastiera memorizzando il risultato nella variabile `somma` e il ciclo termina quando digitiamo `0` come prima cifra.
```
repeat
   read(n);
   somma := somma + n;
until n = 0;
```

4 Programmazione

Un insieme di istruzioni da eseguirsi una di seguito all'altra costituisce un programma e la scrittura di queste istruzioni si chiama programmazione.

Spesso, ed è bene lo sia tutte le volte che è complesso, un programma è suddiviso in tante procedure e funzioni, chiamate anche sottoprogrammi, che vengono richiamate per essere eseguite quando serve.

Abbiamo visto nel precedente Capitolo che alcune procedure e funzioni sono comprese nel pacchetto base di Free Pascal e sono richiamabili senza particolari procedimenti.

Altre le possiamo scrivere noi nello stendere il programma.

Abbiamo anche molte procedure e funzioni raggruppate in librerie aggiuntive al pacchetto base, librerie che in Free Pascal si chiamano `unit`.

Noi stessi possiamo creare librerie per avere a disposizione nostre procedure e funzioni originali anche per altri programmi.

Programmi, procedure e funzioni sono sempre costituiti da una intestazione, da una parte dichiarativa e dal blocco delle istruzioni.

La parte dichiarativa è di fondamentale importanza nel linguaggio Free Pascal e vi si compendia tutta la filosofia della programmazione strutturata: prima di scrivere un programma dobbiamo avere chiaro che cosa vogliamo ottenere in modo da poter dichiarare, ancor prima di scrivere le istruzioni, che cosa ci serve.

Di ciò che abbiamo visto finora la parte dichiarativa deve contenere, nell'ordine:
. preceduti dalla parola chiave `uses` i nomi delle librerie (unit) eventualmente necessarie (non dimenticando di finire la riga con il punto e virgola),
. le definizioni delle costanti, secondo la sintassi vista nel paragrafo 3.2.1,
. le definizioni delle variabili, secondo la sintassi vista nel paragrafo 3.2.2.

4.1 Procedure

La procedura è un sottoprogramma che svolge determinati compiti, tra i quali vi possono essere anche calcoli, ma non restituisce direttamente alcun valore.

La sintassi per dichiarare una procedura è la seguente:

```
procedure <nome>;
   <sezione_dichiarativa>
   begin
       <istruzioni>
   end;
```

L'indentazione è assolutamente facoltativa e serve per fare in modo che ciò che scriviamo sia umanamente ben leggibile e interpretabile; non influisce assolutamente sulla lettura di ciò che scriviamo da parte del compilatore del programma.

Anticipando che l'istruzione `writeln()` serve per scrivere qualche cosa a terminale, la seguente procedura, richiamata in un programma, scrive tre volte la parola Ciao su righe diverse:

```
procedure scrivi;
   const messaggio = 'Ciao';
   var i:  integer;
   begin
       for i := 1 to 3 do writeln(messaggio);
   end;
```

Nella parte dichiarativa abbiamo inizializzato come costante il messaggio da scrivere e abbiamo dichiarato la variabile `i` che ci servirà da contatore per il ciclo `for` e nelle istruzioni abbiamo inserito il ciclo per scrivere tre volte il messaggio.
Nel corso del programma basterà chiamarla scrivendo l'istruzione `scrivi;` per ottenere la scrittura dei messaggi.
La costante e la variabile dichiarate nella sezione dichiarativa della procedura vengono generate in memoria al momento della chiamata nel programma e, immediatamente dopo l'esecuzione della procedura, lo spazio allocato in memoria viene liberato.

4.2 Funzioni

La funzione è un sottoprogramma che ritorna un valore del tipo previsto nell'intestazione e denominato come la funzione.

La sintassi per dichiarare una funzione è la seguente:

```
function <nome> (<parametro>:<tipo>;<parametro>:<tipo>;...)  :<tipo>;
   <sezione_dichiarativa>
   begin
       <istruzioni>
   end;
```

I parametri sono i valori necessari perché la funzione svolga il proprio compito e vanno indicati, nell'ordine previsto e tra parentesi tonde, quando si chiama la funzione.

Tra le istruzioni non potrà mancare quella di assegnazione del valore restituito dalla funzione alla variabile avente lo stesso nome della funzione.

La seguente funzione calcola l'area del triangolo avente una certa base e una certa altezza:

```
function area_triangolo(base: double; altezza: double): double;
   begin
       area_triangolo := base*altezza/2;
   end;
```

Nel corso del programma, richiamandola, per esempio, con l'istruzione `area_triangolo(3,2)`, restituisce il valore 3, che è l'area di un triangolo di base 3 e altezza 2. Richiamata con l'istruzione `area_triangolo(7.5, 13.2)` restituisce il valore 49,5.

4.3 Unit

Le procedure e le funzioni che costruiamo con la sintassi vista nei precedenti paragrafi possono essere inserite come tali nel listato del programma.

Esiste tuttavia la possibilità di raccogliere procedure e funzioni in librerie esterne al programma, in modo che siano utilizzabili in qualsiasi programma, semplicemente dichiarando la volontà di usarle. In Free Pascal queste librerie si chiamano unit.

Possiamo costruire noi stessi delle unit con la sintassi:

```
unit <nome>;
   interface
       <prototipi>
   implementation
       <procedure e funzioni>
end.
```

dove

<prototipi> è l'elenco delle intestazioni delle procedure e delle funzioni che inseriamo nella unit e costituiscono l'<interface>,

<implementation> contiene le procedure e le funzioni costruite con le sintassi viste nei precedenti paragrafi.

Una volta scritto il file di programmazione della unit con un editor di testo, lo salviamo con lo stesso nome dato alla unit e con l'estensione `.pas` e lo compiliamo nello stesso modo con cui si compilano i programmi, come abbiamo visto nel Capitolo 2.

Otteniamo così due nuovi file aventi lo stesso nome del file `.pas`, l'uno con estensione `.o` e l'altro con l'estensione `.ppu`.

Questi due file vanno tenuti a disposizione nella directory dove scriviamo il programma che richiama la unit oppure, meglio, vanno archiviati nella posizione dove verranno altrimenti cercati dal compilatore del programma che richiama la unit:

`/usr/lib/fpc/3.2.0/units/` o similare in Linux e Mac,

`C:\FPC\3.2.0\units` o similare in Windows con installato solo il compilatore,

`C:\lazarus\units` in Windows con installato Lazarus.

Per esempio, Free Pascal non ha una funzione predefinita per calcolare il fattoriale di un numero. Se prevediamo di avere spesso bisogno di utilizzare questa funzione nei nostri programmi possiamo creare una libreria che la contenga e, in previsione di arricchirla con altre funzioni matematiche, chiamiamola matematica.

```
unit matematica;
interface
   function fattoriale(n: integer): extended;
implementation
   function fattoriale(n: integer): extended;
   begin
       if n = 1 then fattoriale := 1
       else fattoriale := n * fattoriale(n-1);
   end;
end.
```

Scrivendo i programmi in cui vogliamo utilizzare questa funzione, dichiariamo l'uso della nostra libreria nella parte dichiarativa con
`uses matematica;`
e potremo richiamare la funzione `fattoriale()` ivi contenuta quante volte vogliamo, passandole un numero intero come parametro tra le parentesi tonde.
Avendo scelto come tipo del risultato `extended`, esso sarà espresso in notazione scientifica con le sole prime 20 cifre esatte.

4.4 Programma

Un programma è il risultato più compiuto dell'attività di programmazione quando questa, attraverso la compilazione, sfocia nella produzione di un file eseguibile.

Nei precedenti tre paragrafi abbiamo visto attività di sotto-programmazione e di supporto alla programmazione, attività che, anche se, come per la unit, le sottoponiamo a una compilazione, non sfociano nella produzione di un file eseguibile.

La sintassi per produrre un programma è la seguente:

```
program <nome>;
    <sezione_dichiarativa>
    begin
        <istruzioni>
    end.
```

Notare come la sintassi sia la stessa prevista per la procedura, con la sola differenza che dopo la parola finale `end` abbiamo un punto fermo (`.`) anziché un punto e virgola (`;`).

5 Interattività con l'utente

Non necessariamente, ma quasi sempre, il funzionamento di un programma prevede interattività con l'utente: normalmente, infatti, un programma per computer prevede l'inserimento e la stampa di messaggi e dati.

L'interfacciamento a questo scopo può avvenire utilizzando il terminale che si apre sullo schermo del computer e la tastiera del computer stesso (in tal caso si parla di programma per console) oppure attraverso una apposita finestra grafica creata sullo schermo del computer ed attrezzata in modo che vi si possa agire anche utilizzando il mouse (in tal caso si parla di programma con GUI, Graphical User Interface).

5.1 Programmi console

Possiamo scriverli con l'editor di Lazarus scegliendo da menu FILE ▷ NUOVO ▷ PROGETTO ▷ PROGRAMMA SEMPLICE o FILE ▷ NUOVO ▷ PROGETTO ▷ PROGRAMMA.

Personalmente, come ho detto nel Capitolo 2, ritengo meglio utilizzare l'editor Geany. Non da scartare l'idea di utilizzare l'editor abbinato al compilatore, che si lancia con il comando `fp`.

5.1.1 Output

Si realizza con le istruzioni `write()` e `writeln()`.

La differenza tra le due sta nel fatto che la prima scrive e non va accapo mentre la seconda scrive e fa un salto a nuova linea.

Tra le parentesi si indica cosa scrivere. Se si omettono le parentesi o non vi si inserisce nulla, nel caso di `writeln` inserito tra altre istruzioni `writeln`, si realizza una riga vuota.

Il cosa scrivere può essere una stringa scritta tra apici semplici, un valore numerico scritto in termini letterali, il valore di una variabile richiamandone il nome, il risultato di una espressione matematica. E' possibile indicare più cose da scrivere sulla stessa riga separandole con una virgola.

Data l'esistenza di una variabile `x` contenente il valore `5` e di una variabile `s` contenente il valore `Vittorio`,
`writeln('Ciao');` scrive Ciao
`writeln(s);` scrive Vittorio
`writeln('Ciao, ', s);` scrive Ciao, Vittorio
`writeln(12);` scrive 12
`writeln(7/2);` scrive 3.5
`writeln(x);` scrive 5
`writeln('x vale:  ', x);` scrive x vale 5
`writeln('3 + 7 fa ', 3+7);` scrive 3 + 7 fa 10

Esiste la possibilità di formattare output numerici indicando gli spazi dedicati per il relativo posizionamento e le cifre decimali da evidenziare, con la sintassi
`writeln(<valore>: <spazi>: <decimali>);`

Le istruzioni consecutive
```
writeln(pi:10:5);
writeln(12.7657485768:10:5);
```
scrivono i numeri così posizionati e dimensionati
␣␣␣3.14159
␣␣12.76575

5.1.2 Input

Per incamerare dati digitati sulla tastiera abbiamo le istruzioni `read()` e `readln()`.

La differenza tra le due sta nel fatto che la prima legge e non va accapo mentre la seconda legge ed effettua un salto a linea nuova.

Tra le parentesi si indica il nome della variabile in cui va inserito il dato.

L'istruzione `readln` seguita direttamente dal punto e virgola si può utilizzare per provocare un'attesa indefinita durante l'esecuzione del programma, in attesa della pressione di un tasto.

Tra le parentesi si possono indicare più nomi di variabili separati da virgola. In tal caso sulla tastiera occorre digitare i relativi valori separati da almeno uno spazio.

All'istruzione `read()` o `readln()` è opportuno far precedere un'istruzione `write()` o `writeln()` contenente la descrizione dell'input richiesto.

5.1.3 Semplici programmi console di esempio

Area e circonferenza di un cerchio

Non ci serve richiamare alcuna libreria. Le variabili che ci interessano sono il raggio del cerchio per l'input, l'area e la circonferenza per l'output e, avendo a che fare con grandezze ragionevolmente limitate esprimenti numeri reali, ci basta siano di tipo double.
Il listato del programma potrebbe essere questo:

```
program cerchio;
var
   raggio, area, circonferenza:  double;
begin
   write('Inserisci il raggio del cerchio:  ');
   readln(raggio);
   area := sqr(raggio) * pi;
   circonferenza := raggio * 2 * pi;
   writeln('area:  ', area:0:3);
   writeln('circonferenza:  ', circonferenza:0:3);
   write('Premi INVIO per terminare');
   readln;
end.
```

I risultati vengono espressi con tre cifre decimali nella posizione del cursore (formattazione `:0:3`).
Le ultime due righe del programma sono necessarie, se usiamo il sistema operativo Windows, per mantenere aperta la finestra del prompt dei comandi fino alla pressione del tasto INVIO. In mancanza di ciò, lanciato il programma richiamando l'eseguibile o cliccandoci sopra, il prompt dei comandi, eseguito il programma, si chiuderebbe senza lasciarci il tempo di vedere i risultati delle elaborazioni.

Permutazioni

Nel calcolo combinatorio le permutazioni corrispondono al fattoriale del numero degli elementi da permutare.

Se abbiamo costruito la unit matematica come previsto nel Paragrafo 4.3 e l'abbiamo archiviata a dovere, il listato del programma che calcola le permutazioni possibili con n elementi potrebbe essere il seguente:

```
program permutazioni;
uses matematica;
var n:  integer;
begin
    write('Numero degli elementi:  ');
    readln(n);
    writeln(fattoriale(n):0:0);
    write('Premi INVIO per terminare');
    readln;
end.
```

La formattazione del risultato (`:0:0`) è per evitare l'esposizione del risultato stesso in notazione scientifica come avverrebbe, dato il tipo attribuitogli nella unit `matematica`.

Se non abbiamo a disposizione la unit, possiamo inserire la funzione che calcola il fattoriale nel programma:

```
program permutazioni;
var n:  integer;
function fattoriale(n:  integer):  extended;
   begin
       if n = 1 then fattoriale := 1
       else fattoriale := n * fattoriale(n-1);
   end;
begin
   write('Numero degli elementi:  ');
   readln(n);
   writeln(fattoriale(n):0:0);
   writeln
   write('Premi INVIO per terminare');
   readln;
end.
```

Notare come il sottoprogramma (la funzione) che calcola il fattoriale preceda il blocco del vero e proprio programma, che deve sempre essere l'ultimo scritto.

Saluto personalizzato

Qui il programma chiede il nome all'utente per poterlo salutare come si deve.

```
program saluto;
var nome:  string;
begin
   write('Come ti chiami?  ');
```

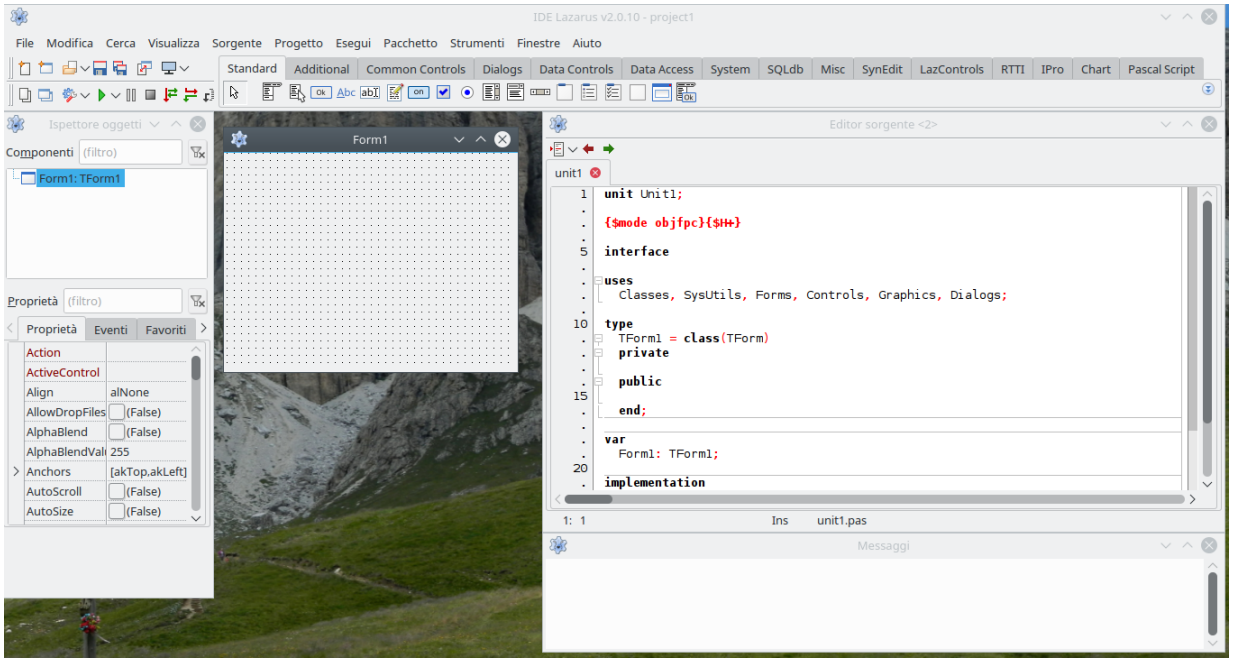
```
    readln(nome);
    writeln('Ciao, caro ', nome, '!');
    write('Premi INVIO per finire');
    readln;
end.
```

5.2 Programmi con GUI

Per realizzare programmi con interfaccia utente grafica usiamo Lazarus.

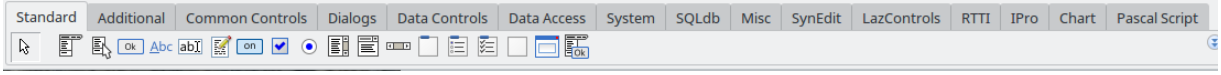
Al suo lancio, Lazarus si appresta a questo compito, corrispondente alla scelta di menu FILE ▷ NUOVO ▷ PROGETTO ▷ APPLICAZIONE.

L'area di lavoro si presenta così



In alto abbiamo innanzi tutto la barra dei menu per la scelta delle varie cose che vogliamo fare. Appena sotto, sulla sinistra, abbiamo due piccole barre di strumenti sovrapposte

che facilitano l'accesso ai compiti più ricorrenti attraverso click su icone. Passando il mouse sulle icone stesse siamo informati sullo strumento che rappresentano.
Sulla destra di queste due barre abbiamo la zona dedicata alla scelta dei componenti l'interfaccia che vogliamo costruire



I componenti, altrimenti detti widgets, sono raggruppati in quindici schede.
Per default è aperta la prima, denominata STANDARD, che contiene i componenti più comuni, quelli che sicuramente bastano e avanzano per un principiante.
Sotto la linea per la scelta della scheda si allineano le icone dei componenti appartenenti alla scheda aperta. Le icone sono autoesplicative, comunque, passando il mouse su di esse, ne vediamo la definizione.

Se esploriamo le varie schede ci rendiamo conto di quanto sia vasta la strumentazione di Lazarus.

Sotto questa zona sono collocate le finestre di lavoro.

Sulla sinistra abbiamo l'ISPETTORE OGGETTI, suddiviso in due finestre: quella in alto presenta l'elenco dei componenti impegnati nella nostra interfaccia in modo che li possiamo tenere sotto controllo e selezionarli cliccando sopra il loro nome con il mouse. Quella in basso (scheda PROPRIETÀ) elenca le proprietà del componente selezionato in una struttura atta ad intervenire per modificarle a nostro piacimento. Vi possiamo aprire anche la scheda EVENTI.

Sulla destra abbiamo l'EDITOR SORGENTE, nel quale lo stesso Lazarus inserisce il codice corrispondente alla realizzazione dell'interfaccia con le componenti scelte e il programmatore è chiamato ad inserire il codice delle procedure e delle funzioni per svolgere i compiti che gli eventi captati dall'interfaccia richiedono.

Al centro abbiamo il componente base dell'interfaccia grafica, il Form, cioè la finestra in cui inserire gli altri vari componenti con cui costruire l'interfaccia stessa. Possiamo dimensionarla a piacimento agendo per trascinamento con il mouse sui bordi e sugli angoli.

Con questa strumentazione, se abbiamo padronanza delle basi del linguaggio Free Pascal, ci basta acquisire dimestichezza con i meccanismi di Lazarus per realizzare interfacce grafiche anche impegnative. Questi meccanismi è difficile spiegarli in un manuale ma, provando e riprovando con pazienza e grazie all'intuitività su cui si basa Lazarus, non occorre molto per arrivare a destreggiarsi.

Alla base di tutto sta il concetto che i componenti dell'interfaccia che andiamo a costruire sono informaticamente degli oggetti:
. dotati di proprietà, alle alle quali possiamo accedere con istruzioni composte dal nome dell'oggetto seguito, separato da un punto (`.`), dal nome della proprietà,
. collegabili ad un evento da cui si possano far dipendere conseguenze: che ci si passi sopra con il mouse, che ci si clicchi sopra, ecc.

5.2.1 Output

Il più comodo componente per l'output in una GUI è la Label (tipo `TLabel`), che automaticamente, in ordine di inserimento nel Form, prende i nomi `Label1`, `Label2`, ecc.

Essa si presta per scrivere nel form, in posizioni volute, titoli, istruzioni, ecc. oppure per scrivere il risultato di elaborazioni.

La proprietà che consente di inserire del testo nella Label è `Caption`.

Con la sintassi

```
labelx.Caption := <valore>;
```

inseriamo nella Label numero `x` un valore, come se la Labelx fosse una variabile del linguaggio Pascal di base.
Questo valore può essere solo una stringa o una concatenazione di stringhe (<stringa> + <stringa> +...).
Valori numerici si inseriscono convertiti con `inttostr()` o `floattostr()` a seconda del tipo.

Esiste una funzione per formattare un numero decimale convertito in stringa:
`floattostrf(<valore>, <formato>, <precisione>, <decimali>)`
dove
<valore> è il numero o l'espressione da convertire,
<formato> è il tipo di conversione: i più interessanti sono
`ffgeneral` per ottenere un numero non in notazione scientifica,
`ffnumber` per ottenere lo stesso con la separazione delle migliaia,
<precisione> è il numero di cifre significative,
<decimali> è il numero di cifre decimali (con arrotondamento dell'ultima).
`floattostrf(<valore>, ffnumber, 20, 3)`
scrive <valore> in formato con separatori delle migliaia (,) e 3 decimali.

5.2.2 Input

Il classico componente per acquisire dati in una GUI è l'Edit (tipo `TEdit`), che automaticamente, in ordine di inserimento nel Form, prende i nomi di `Edit1`, `Edit2`, ecc.

Si tratta di una finestrella nella quale si scrive un dato digitandolo sulla tastiera. La proprietà che registra ciò che si scrive nella finestrella è `Text`.

Con la sintassi

<variabile> `:= editx.Text;`

acquisiamo in una variabile il valore inserito nella finestrella numero `x`, che è una stringa.

Se la variabile è di tipo numerico occorre usare le funzioni di conversione `strtoint()` o `strtofloat()` a seconda del tipo.

<variabile> `:= strtoint(editx.Text);` inserisce quanto letto come numero intero,

<variabile> `:= strtofloat(editx.Text);` inserisce quanto letto come numero decimale.

5.2.3 Semplici programmi con GUI di esempio

Cominciamo a vedere come si imposta un progetto con Lazarus.

Creiamo innanzitutto una cartella, dove ci è più comodo, destinata a contenere i file del progetto e la chiamiamo `mio_progetto` (nel caso specifico al posto di mio_progetto mettiamo un nome evocativo del contenuto del progetto).

Lanciamo Lazarus.

Lazarus ha l'abitudine di aprirsi sull'ultimo progetto sviluppato. Per affrontare un nuovo progetto ricorriamo al menu FILE ▷ NUOVO ▷ PROGETTO ▷ APPLICAZIONE. Ci troviamo così di fronte l'area di lavoro illustrata a pagina 17.

Da menu scegliamo FILE ▷ SALVA COME... e salviamo il primo file che ci viene proposto, quello con estensione `.lpi`, dandogli il nome che abbiamo assegnato al progetto (`mio_progetto` o quello più evocativo che abbiamo scelto) e mantenendo l'estensione `.lpi`. Salviamo poi il secondo file che ci viene proposto, denominato `unit1.pas` così com'è.

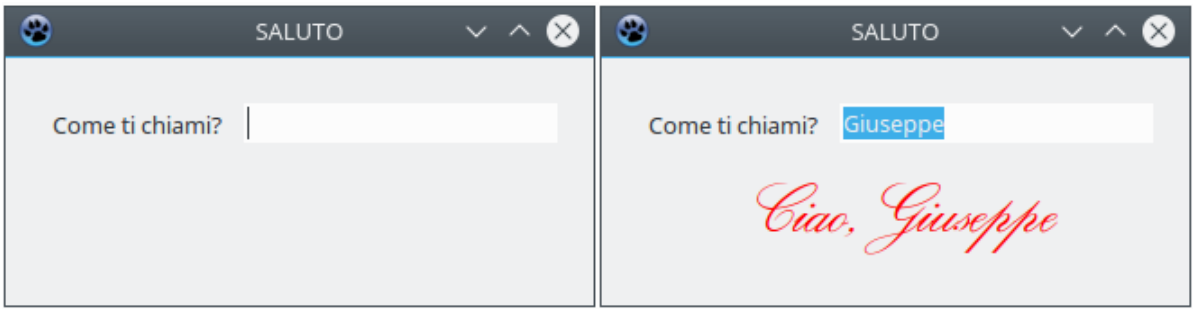
Ora siamo pronti a sviluppare il nostro progetto raccogliendo il nostro lavoro in maniera ordinata nella cartella dedicata.

Nel corso del lavoro è bene salvare ricorrentemente ciò che abbiamo fatto ricorrendo al menu FILE ▷ SALVA TUTTO.

Saluto personalizzato

E' la versione GUI dell'ultimo esempio svolto nel Paragrafo dedicato ai programmi console: il computer chiede all'utente il suo nome e lo saluta.

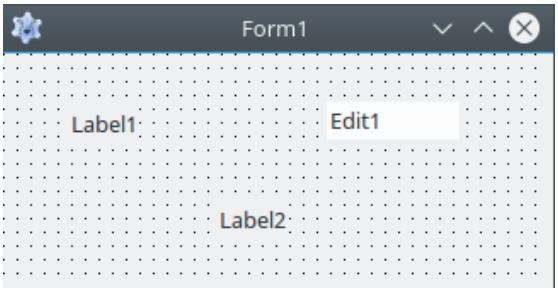
Il programma si apre come da schermata di sinistra. Premendo INVIO dopo aver inserito il nome nella finestrella, digitandolo sulla tastiera, la schermata diventa quella di destra. Il programma si chiude cliccando sulla `X` in alto a destra.



Impostato il progetto come visto sopra, prima cosa è disegnare l'interfaccia.

Nell'esemplare di questa che troviamo al centro dell'area di lavoro inseriamo le componenti che ci servono: una finestrella Edit (icona) e due Label (icona). Selezionata l'icona nella barra dei componenti clicchiamo sul form nella posizione dove inserire.

Il form diventa



Diamo un titolo al form:
. selezioniamo il form cliccando sul suo nome nell'elenco dei componenti dell'ISPETTORE OGGETTI oppure cliccando in una zona libera del form stesso,
. scorriamo nella zona delle proprietà dell'ISPETTORE OGGETTI fino a trovare la proprietà CAPTION e nella finestrella a destra del nome scriviamo SALUTO.

Ora dedichiamoci ai particolari dell'interfaccia.

Utilizziamo la Label1 per il messaggio di richiesta del nome dell'utente che vuole farsi salutare:
. selezioniamo la Label1 cliccando sul suo nome nell'elenco dei componenti dell'ISPETTORE OGGETTI oppure cliccando su di essa nel form,
. nella zona delle proprietà dell'ISPETTORE OGGETTI selezioniamo la proprietà CAPTION e nella finestrella a destra del nome scriviamo Come ti chiami?

Utilizziamo la finestrella Edit1 per far immettere il nome all'utente:
. selezioniamo la Edit1 cliccando sul suo nome nell'elenco dei componenti dell'ISPETTORE OGGETTI oppure cliccando su di essa nel form,
. scorriamo nella zona delle proprietà dell'ISPETTORE OGGETTI fino a trovare la proprietà TEXT e cancelliamo ciò che c'è scritto nella finestrella a destra del nome,
. trascinando il mouse con premuto il tasto sinistro sui bordi sinistro e destro del componente diamogli una dimensione acconcia (per esempio che vada dal termine della Label1 fino al bordo del form).

Sempre lavorando di trascinamento con il mouse posizioniamo Label1 e Edit1 in modo allineato.

Ci rimane la Label2, che riserviamo all'inserimento del saluto:
. selezioniamo la Label2 cliccando sul suo nome nell'elenco dei componenti dell'ISPETTORE OGGETTI oppure cliccando su di essa nel form,
. selezioniamo la proprietà AUTOSIZE nella zona delle proprietà dell'ISPETTORE OGGETTI e disattiviamo l'opzione TRUE nella finestrella a destra del nome in modo che diventi FALSE,
. lavorando di trascinamento del mouse sui bordi del componente allarghiamo la zona della Label in modo che occupi tutto lo spazio restante del form sotto i componenti sistemati prima,
. nella zona delle proprietà dell'ISPETTORE OGGETTI selezioniamo la proprietà CAPTION e cancelliamo il contenuto della finestrella a destra del nome,
. nella zona delle proprietà dell'ISPETTORE OGGETTI selezioniamo la proprietà ALIGNMENT e cliccando nella finestrella a destra del nome, scegliamo TACENTER,
. infine scegliamo un font per la scritta del saluto: sempre nella zona delle proprietà dell'ISPETTORE OGGETTI scorriamo fino a selezionare la proprietà FONT e apriamo il relativo menu cliccando sul pulsante con tre puntini a destra del nome; nella finestra di dialogo scegliamo font e dimensioni (nel mio caso ho scelto RegencyScriptFLF, che ho la fortuna di avere sul computer dove ho lavorato, dimensione 32). Se apriamo le sotto proprietà della proprietà FONT cliccando sulla freccina a sinistra del nome, possiamo selezionare COLOR e, cliccando sul pulsante con tre puntini a destra del nome, aprire la finestra di dialogo per scegliere il colore con cui scrivere il saluto (nel mio caso ho scelto un rosso vivo).

Abbiamo così realizzato visivamente la parte grafica dell'interfaccia utente e Lazarus si è occupato di tradurla in istruzioni leggibili dal compilatore.

A questo punto dobbiamo inserire il codice per ciò che si deve realizzare con la parte grafica.

Nel nostro caso si tratta di acquisire ciò che l'utente scrive nel componente Edit1 e utilizzarlo per scrivere la stringa del saluto da inserire nel componente Label2 e prevedere un evento che dia avvio a questa azione: nel nostro caso abbiamo scelto che l'evento sia la pressione di INVIO dopo che è stato scritto il nome della persona da salutare nel componente Edit1 (Lazarus definisce questo evento EDITINGDONE).

Selezioniamo allora, nella maniera che abbiamo ormai imparato, il componente in cui si deve verificare l'evento, che è Edit1, e nella finestrella in basso dell'ISPETTORE OGGETTI apriamo la scheda EVENTI: scorriamo fino a trovare ONEDITINGDONE, selezioniamolo cliccandoci sopra e clicchiamo sul pulsante con tre puntini che compare a destra del nome.

Veniamo così posizionati con il cursore nella finestra dell'editor nel punto in cui inserire le istruzioni nel codice della procedura relativa all'evento selezionato, già predisposta da Lazarus: tra begin e end, dove lampeggia il cursore, scriviamo

```
label2.Caption:='Ciao, ' + edit1.Text;
```

La code completion di Lazarus ci aiuta a farlo.

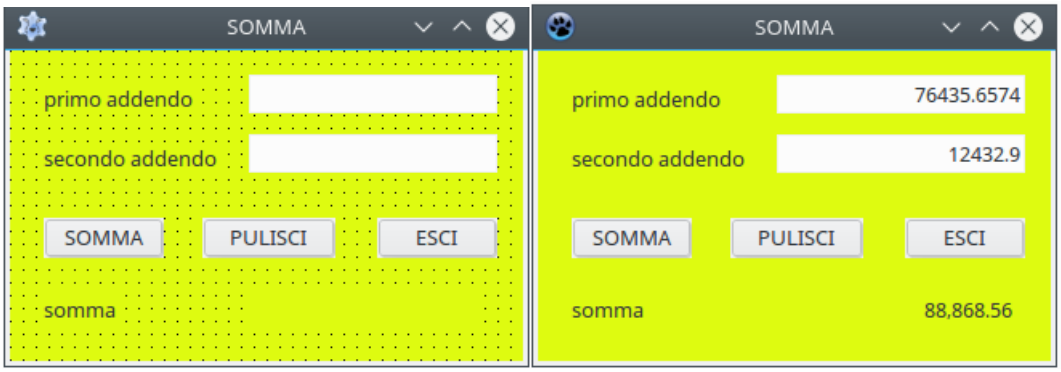
Da menu FILE ▷ SALVA TUTTO salviamo il nostro lavoro.

Da menu ESEGUI ▷ COMPILA compiliamo e produciamo l'eseguibile, che troveremo nella cartella dove abbiamo salvato il progetto: è il file che porta il nome del progetto senza alcuna estensione (o estensione `.exe` se siamo in Windows).

Somma

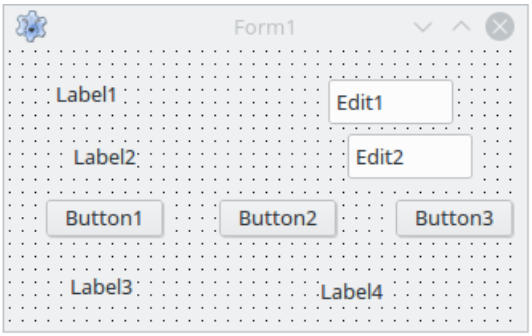
Un programma facile facile che esegue semplicemente l'addizione di due addendi, nel quale vediamo applicate un po' di cose utili.

La schermata del programma è questa



A sinistra abbiamo la schermata di avvio e a destra la schermata dopo che è stata eseguita l'addizione tra i due addendi inseriti nelle apposite finestrelle: il risultato è espresso con un formato che evidenzia le migliaia e approssima a due cifre decimali.

I componenti sono i seguenti



Rispetto all'esempio precedente abbiamo la novità dei pulsanti (Button nella terminologia di Lazarus), che corrispondono al componente con l'icona Ok, e del colore di sfondo del form.

Il colore di sfondo del form si ottiene selezionando il form e, scorrendo le proprietà, scegliendo Color: agendo sul menu che si apre cliccando sul pulsante con i tre puntini a destra del nome della proprietà selezionata scegliamo il colore che più ci piace.

Con le tecniche viste nel precedente esempio dedichiamo Label1 alla descrizione 'primo addendo' e Edit1 al suo inserimento, dedichiamo Label2 alla descrizione 'secondo addendo' e Edit2 al suo inserimento, dedichiamo Label3 alla descrizione della riga del risultato 'somma' e Label4 a mostrare il risultato della somma.

I pulsanti li dedichiamo a tre compiti diversi: il primo ad eseguire la somma, il secondo a ripulire l'interfaccia dai dati di una somma precedente, il terzo ad uscire dal programma. Per scrivere sul pulsante un titolo evocativo del compito ('somma', 'pulisci' ed 'esci') selezioniamo il pulsante e nelle proprietà, scriviamo il titolo nella finestrella della proprietà CAPTION.

Ora dobbiamo far corrispondere ai tre pulsanti le procedure per i compiti assegnati.

Selezionato Button1, apriamo la scheda Eventi dell'Ispettore Oggetti e selezioniamo l'evento OnClick. Nell'editor facciamo in modo che la procedura abbozzata da Lazarus, inserendo i dati in rosso, diventi la seguente:

```
procedure TForm1.Button1Click(Sender:  TObject);
   var a,b:  double;
begin
   a:=strtofloat(edit1.Text);
   b:=strtofloat(edit2.Text);
   label4.Caption:=floattostrf(a + b, ffnumber,20,2);
end;
```

Selezionato Button2, apriamo la scheda Eventi dell'Ispettore Oggetti e selezioniamo l'evento OnClick. Nell'editor facciamo in modo che la procedura abbozzata da Lazarus, inserendo i dati in rosso, diventi la seguente:

```
procedure TForm1.Button2Click(Sender:  TObject);
begin
   edit1.Text:='';
   edit2.Text:='';
   label4.Caption:='';
end;
```

Selezionato Button3, apriamo la scheda Eventi dell'Ispettore Oggetti e selezioniamo l'evento OnClick. Nell'editor facciamo in modo che la procedura abbozzata da Lazarus, inserendo i dati in rosso, diventi la seguente:

```
procedure TForm1.Button3Click(Sender: TObject);
begin
   close;
end;
```

Salviamo tutto e compiliamo.

6 Abbellimento della console

Quello che oggi chiamiamo Terminale in Linux e Mac OS X e Prompt dei comandi in Windows, in cui utilizziamo i così detti programmi console, è, in realtà, un emulatore dello schermo dei veri e propri terminali con cui si lavorava con i mainframe e che è poi diventato lo schermo dei primi personal computer, quelli nei quali non c'erano ancora le interfacce grafiche cui siamo abituati ormai da molti anni (X Window System del 1984 entrato in Mac e Linux, Windows del 1985 come ambiente a finestre del sistema operativo MS-DOS, poi diventato lui stesso sistema operativo).

Su quegli schermi si collocavano i caratteri per interfacciarsi con l'utenza su 80 colonne e 24 righe, dimensioni consacrate nel 1971 con il terminale IBM 3270. Successivamente la stessa IBM

produsse terminali a 25 righe. Sono praticamente queste le dimensioni degli attuali emulatori di terminale che abbiamo sui nostri computer[3].

A quei tempi si trattava di schermi a tubo catodico e ancor oggi la unit del linguaggio Pascal concepita per abbellire lo schermo del terminale su cui si lavorava si chiama `crt`, che è l'acronimo di Cathodic Ray Tube.

Oggi possiamo utilizzare questa unit per abbellire l'emulatore del terminale per il quale era stata concepita.

Per utilizzare la unit dobbiamo importarla con `uses crt;`.

Le procedure che ci mette a disposizione questa unit, prescindendo da quelle ormai anacronistiche[4], sono le seguenti.

Controllo del display

`textbackground(<colore>)` imposta il colore di fondo;

<colore> si indica con

`0`	per il nero
`1`	per il blu
`2`	per il verde
`3`	per il turchese (cyan)
`4`	per il rosso
`5`	per il magenta
`6`	per il marrone
`7`	per il grigio chiaro

`clrscr` ripulisce lo schermo e estende il colore di fondo a tutto lo schermo;
`textcolor(<colore>)` imposta il colore del carattere;

<colore> si indica con

`0`	per il nero	`8`	per il grigio chiaro
`1`	per il blu	`9`	per il blu chiaro
`2`	per il verde	`10`	per il verde chiaro
`3`	per il turchese (cyan)	`11`	per il turchese chiaro
`4`	per il rosso	`12`	per il rosso chiaro
`5`	per il magenta	`13`	per il magenta chiaro
`6`	per il marrone	`14`	per il giallo
`7`	per il grigio chiaro	`15`	per il bianco

Controllo del cursore

La posizione del cursore è determinata dalle coordinate `x`, per la posizione sulle colonne, e `y` per la posizione sulle righe. Ricordo che, in eredità dei vecchi terminali, le colonne sono 80 e le righe 25.

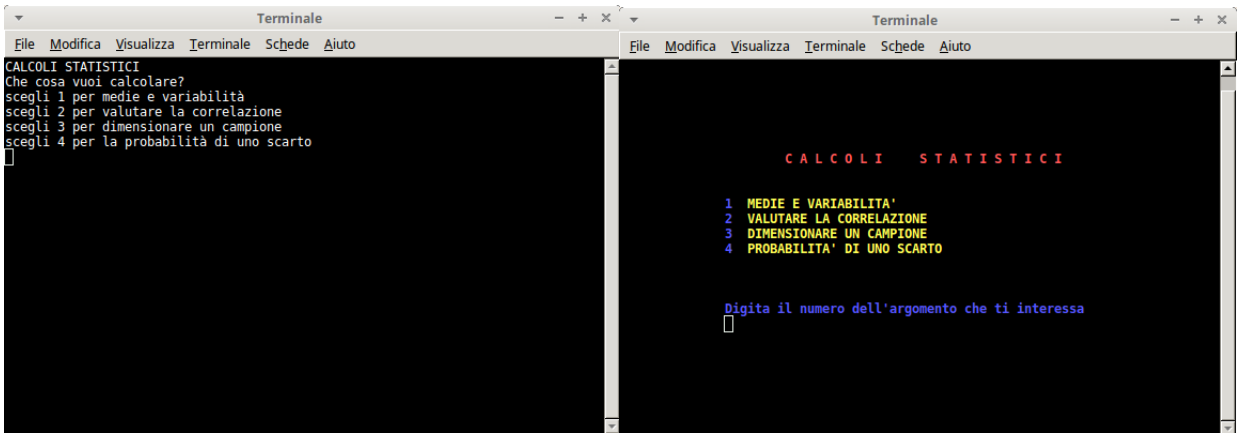
`gotoxy(x, y)` posiziona il cursore sulla colonna `x` e sulla riga `y`,
`wherex` ritorna il numero della colonna su cui si trova il cursore,
`wherey` ritorna il numero della riga su cui si trova il cursore.

Può essere quanto serve per rendere un po' più gradevole il freddo emulatore di terminale.

[3] A chi interessi conoscere le origini di queste dimensioni posso dire che 80 erano le colonne della prima scheda perforata standard IBM del 1928, che tuttavia aveva solo 12 righe. Forse il numero delle righe del terminale era inizialmente semplicemente il doppio di queste, per rendere il rettangolo di lavoro meno schiacciato e dare allo schermo una forma più da schermo che da scheda perforata.

[4] Per esempio, esiste la procedura window(x1, y1, x2, y2), con x minore di 80 e y minore di 25, per disegnare una finestra più piccola dello schermo intero ai tempi del vero e proprio terminale. Oggi questa procedura disegnerebbe una finestra all'interno dell'emulatore del terminale che, a sua volta, è una finestra minore dello schermo e ciò ritengo non possa interessare più nessuno.

Per evidenziare la differenza tra un terminale non abbellito e un terminale abbellito presento il confronto tra due menu di un ipotetico programma di calcoli statistici, il primo (sulla sinistra) risultato di una procedura scritta con il normale linguaggio, il secondo (sulla destra) risultato di una procedura scritta utilizzando la unit `crt`.



Il codice in linguaggio normale è il seguente:

```
procedure menu;
    begin
        writeln('CALCOLI STATISTICI');
        writeln('Che cosa vuoi calcolare?');
        writeln('scegli 1 per medie e variabilità');
        writeln('scegli 2 per valutare la correlazione');
        writeln('scegli 3 per dimensionare un campione');
        writeln('scegli 4 per la probabilità di uno scarto');
    end;
```

Il codice utilizzando la unit `crt` è il seguente:

```
uses crt;
procedure menu;
    begin
        textbackground(0);
        clrscr;
        gotoxy(23,7); textcolor(12);
        write('C A L C O L I     S T A T I S T I C I');
        gotoxy(15,10); textcolor(9);
        write('1');
        gotoxy(18,10); textcolor(14);
        write('MEDIE E VARIABILITA'');
        gotoxy(15,11); textcolor(9);
        write('2');
        gotoxy(18,11); textcolor(14);
        write('VALUTARE LA CORRELAZIONE');
        gotoxy(15,12); textcolor(9);
        write('3');
        gotoxy(18,12); textcolor(14);
        write('DIMENSIONARE UN CAMPIONE');
        gotoxy(15,13); textcolor(9);
        write('4');
        gotoxy(18,13); textcolor(14);
        write('PROBABILITA'' DI UNO SCARTO');
        gotoxy(15,17); textcolor(9);
        write('Digita il numero dell''argomento che ti interessa');
        gotoxy(15,18);
    end;
```

7 Lavorare con i file

Nella documentazione ufficiale si trovano modalità di trattare file con Free Pascal più sofisticate di quella che presento qui, dove mi propongo semplicemente di descrivere il meccanismo di base.

Variabile di tipo file

La prima cosa da fare è creare una variabile di tipo file su cui lavorare.

Il tipo è quello dei dati che il file contiene o è destinato a contenere.

Nella sezione dichiarativa del programma la sintassi è la seguente:

`var` <nome>`:` <tipo_file>`;`

dove

<nome> è il nome della variabile: in genere, per brevità, si sceglie `f`,

<tipo_file> può essere espresso con la sintassi

`file of` <tipo_dato> dove <tipo_dato> è uno di quelli visti nel paragrafo 3.1,

oppure, se si tratta di un comune file di testo, con la parola chiave `textfile`.

Ad esempio:

`var f: file of char`; crea la variabile file `f` per dati di tipo `char`,

`var f: file of integer`; crea la variabile file `f` per dati di tipo numero intero,

`var f: textfile`; crea la variabile file f per file di testo.

Agganciamento della variabile di tipo file a un file sul disco del computer

Nel corpo del programma, con l'istruzione

`assign(`<nome>`,` <percorso_al_file_su_disco>`);`

assegniamo alla variabile di tipo file (per semplicità `f`), il file su cui lavorare, indicandone il percorso nel file system.

Avendo preventivamente dichiarato, nella sezione dichiarativa

```
var f: textfile;,
assign(f, '/home/vittorio/Documenti/prova.txt');
```

nel corpo del programma assegna a questa variabile il file `prova.txt` che si trova all'indirizzo indicato nel mio file system Linux. Stesso stile in Mac. In Windows troveremo percorsi del tipo `'C:\.....'`.

E' molto importante che il file indicato contenga dati del tipo assegnato alla variabile file. In proposito rammento che per scrivere testi non va scelto il tipo `char`, a meno che vogliamo memorizzare di volta in volta singoli caratteri: dobbiamo invece scegliere `textfile`, che ci consente di memorizzare parole e frasi.

Creazione del file

Nel corpo del programma, dopo l'istruzione `assign`, con l'istruzione

`rewrite(`<nome>`,` <percorso_al_file_su_disco>`);`

predisponiamo il file indicato nel percorso alla riscrittura, nel senso che, se il file esiste, viene svuotato per essere riscritto eliminando il contenuto precedente e, se il file non esiste, viene creato vuoto di contenuto.

Aggiornamento del file

Nel corpo del programma, dopo l'istruzione `assign`, con l'istruzione

`append(`<nome>`,` <percorso_al_file_su_disco>`);`

predisponiamo il file indicato nel percorso ad avere aggiunti altri elementi al suo contenuto.

Lettura del file

Nel corpo del programma, dopo l'istruzione `assign`, con l'istruzione
`reset(<nome>, <percorso_al_file_su_disco>);`
predisponiamo il file indicato nel percorso ad essere letto.

Azioni sul file

La scrittura sul file predisposto alla scrittura o all'aggiornamento avviene con l'istruzione
`writeln(<nome>, <testo_tra_apici>);`
Con l'istruzione
`writeln(f, 'Sono finalmente riuscito a scrivere su file con Free Pascal');`
scriviamo la frase indicata nella variabile file `f`.

Non vi sono limitazioni al numero dei caratteri che compongono la frase (nonostante la frase sia scritta tra apici non è di tipo `string`).

Affinché la frase raggiunga effettivamente il file agganciato alla variabile file `f` occorre far seguire l'istruzione
`close(f);`

La lettura dal file predisposto ad essere letto avviene con l'istruzione
`readln(<nome>, <variabile>);`
che legge una riga del file fino ad `eol` (end of line).

Per leggere tutto il file occorre instaurare un ciclo fino ad `eof` (end of file), del tipo
`repeat readln(<nome>, <variabile>) until eof(<nome>);`
Purtroppo la <variabile> in cui viene collocato ciò che si legge deve essere preventivamente dichiarata di tipo `string` e soffre della limitazione di 255 caratteri.
Questa limitazione esclude che Free Pascal possa essere affidabile per leggere file di testo.