1.1 PLIANT

1995 -

Pliant, flessibile in italiano, è una linguaggio creato da Hubert Tonneau nel 1995 ed arricchito nel tempo di funzionalità che gli hanno permesso di diventare un ambiente con caratteristiche di un Sistema Operativo (FullPliant). Pliant ha lo scopo di generare un eseguibile efficiente quanto quello prodotto dal C.

La documentazione, in particolare quella relativa alle nuove evoluzioni, è poco strutturata e parzialmente lacunosa.

Pliant è un linguaggio fortemente tipizzato che prevede una vasta quantità di tipi di dato, suddivisi in grandi classi: numerici, booleani, testo, insiemi e vari tipi caratteristici di vari sottosistemi implementati; inoltre l'utilizzatore può creare i suoi tipi di dato.

La sintassi della dichiarazioni di variabili è: `var` *`tipo variabile`* con un'eventuale assegnazione di valore tramite `:=`. Esiste anche una non ben documentata `ovar`, utilizzata per dichiarare le variabili nell'interfaccia grafica .

I tipi numerici, a parte i generici `Int`, `Float` e `Intn`, questo per numeri interi illimitati, individuano anche la dimensione di memoria ed eventualmente la struttura interna della rappresentazione numerica sul processore: a lato un esempio di alcuni tipi numerici.

`Int`	Intero 32 o 64 bit secondo il processore
`uInt`	Intero non segnato
`Int8`	Intero 8 bit
`Intn`	Interi illimitati
`uInt_hi`	Intero high Indian
`uInt_li`	Intero low Indian

Nell'assegnare un valore numerico ad una variabile si può usare, oltre alla notazione normale anche la binaria e decimale: ad esempio: `99`, `99n`, `1100011b` e `63h`.

Il tipo di dati testo ha gli usuali metodi di estrazione, ricerca e sostituzione di stringhe; è presente il metodo `parse` un metodo relativamente povero, per estrarre delle sottostringhe, ad esempio per portare un numero in una variabile numerica, o per verificarne la struttura:

```
function Parse s -> r  # verify mail address (roughly)
  arg Str s; arg Bool r # Booleans default seems true
  if not (s parse any "@" any "." any)
     r := not r
console (Parse "rukago@condor.it" ) eol                    "true"
console (Parse "abc12puf@condor") eol                      "false"
```

I caratteri speciali sono individuati da un nome incluso fra parentesi quadre, ad esempio `"linea terminata[cr][lf]"`.

E' presente inoltre un metodo curioso `option`, che permette di estrarre da una stringa contenente coppie di nome-valore:

```
var Str s := "nome [dq]Ross[dq] count 3 country [dq]Italy[dq]"
console (s option "name" Str) eol # ([dq] è "
```

Infine il metodo `string` permette di ottenere una rappresentazione stampabile degli oggetti:

```
console string:datetime eol
```

Le funzioni ammettono la ricorsione, la sintassi della dichiarazione è:

`function` *`nomefunzione`* [*`param`*$_1$ `...`] `->` *`risultato`*

ad esempio:

```
function serieArit n -> s
  arg Int n s
  s := n*(n + 1)\2
```

Si noti che i parametri della funzione sono dichiarati tramite `arg` *`tipo nome ...`* (`arg_rw` se il parametro è passato per referenza); inoltre dall'esempio emerge che il corpo della funzione è individuato dall'indentazione, indentazione che è anche usata sia per i blocchi di istruzioni dipendenti da istruzioni di controllo, sia per specificare proprietà ed eventuali eventi degli oggetti dell'interfaccia grafica.
Il richiamo di una funzione è (*`funzione listaparametri`*), nel caso di un solo parametro è possibile scrivere *`funzione`*`:`*`parametro`*, ciò è dovuto alla equivalenza sintattica di `x:y` e `(x y)`, così sono valide entrambe le scritture: `(serieArit 15)` e `serieArit:15`.
Infine `method` è la dichiarazione di funzioni con i parametri prefissi:

```
method n serieArit -> s
  ...
console 10:serieArit eol
console (10 serieArit) eol
```

Nelle strutture di controllo `if` può essere seguito da un numero variabile di `eif` ((else if) ed un finale `else`.
Esistono `for` e `while` e `shunt` una funzione che di fatto è una estensione dell'assegnazione condizionale:

```
r:= (shunt (s parse (i)) "Uns. Number" (s parse (j)) "Unknown")
```

Inoltre `part` *`label`* `...` `loop` *`label`* che implementa una interazione indefinita, dalla quale si può uscire con `leave` *`label`*.
Set è un insieme di tipi di dati caratterizzati dal poter accedere al singolo elemento con o senza chiave e, in questo caso con chiave numerica o qualsivoglia:

- `List` un insieme di dati, accesso solo sequenziale,
- `Array` la chiave di accesso è numerica da 0,
- `Dictionary` dati individuati da una chiave, le chiavi possono essere duplicate,
- `Index` come `Dictionary`, ma i dati sono percorribili in modo ordinato.

Ovviamente sono presenti metodi per percorrere e manipolare queste strutture.

```
function Random i j -> k
  arg Int i j k
  var Float randseed := 10/j
  k := (cast (10^i*randseed-0.5)Int) % 10
function loop i
  arg Int i
  for (var Int j) 1 11
    console (Random j i) " "
console 10/31 eol
loop:31 ; console eol
console 10/97 eol
(loop 97) ; console eol
```

Output:

```
0.322580645161
3 2 2 5 8 0 6 4 5 9 6
0.103092783505
1 0 3 0 9 2 7 8 3 7 7
```

Le discrepanze dipendono dall'imprecisione intrinseca dei numeri floating point.

Sorgente 1.1-1 Generazione di numeri casuali

La dichiarazione di variabili o funzioni deve precedere il loro utilizzo, ciò è non agevole nel caso di programmi con interfaccia grafica.
Pliant integra anche la gestione di Data Base di tipo gerarchico, le tabelle sono descritte non sono altro che un tipo dell'utente:

```
type Article
  field Str descr
  field Float price
  ...
```

I dati sono memorizzati in formato XML o in PML, una variante di XML proprietaria con codifica binaria.

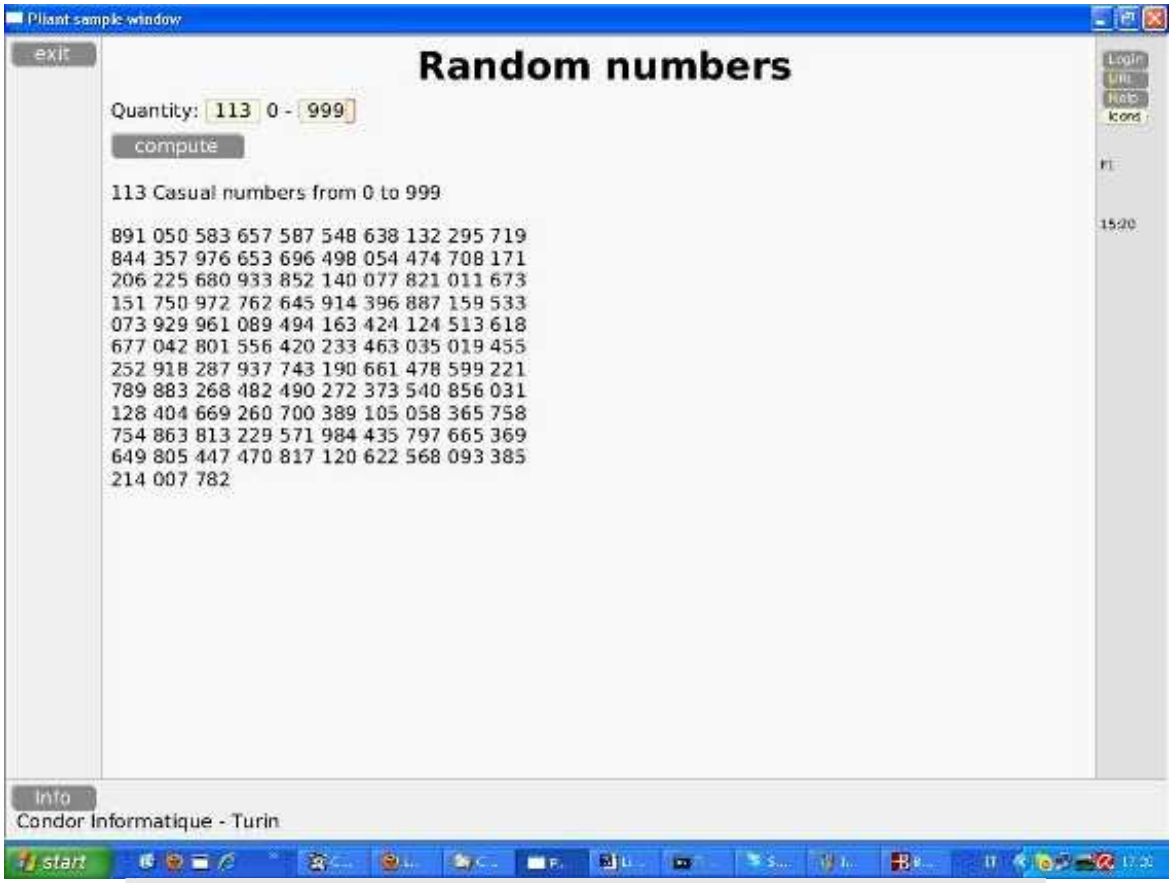
Pliant ha funzioni e tipi di dato per gestire la memoria, gestire i files, la concorrenzialità ed interagire con il Sistema Operativo. Alcune parti sono ancora embrionali, come la libreria per la grafica vettoriale.

FullPliant è un ambiente operativo per eseguire applicazioni Pliant, funziona sia come Server sia come browser, rendendo possibile l'esecuzione remota di programmi Pliant basati sulla sua interfaccia grafica (Pliant UI), in cui lo schermo, la cui geometria è relativamente semplice, è diviso in `window`(s), qualificate da `top`, `left`, `main`, e `bottom`; all'interno delle `window`(s) trovano spazio le `section`(s) per gli oggetti grafici e i `para` per i testi con varie opzioni stilistiche (`bold`, `italic`, `fixed` e `style`); all'interno delle windows l'oggetto `table` aiuta ad organizzare gli spazi. Gli oggetti sensibili sono `link`, `button`, `input` e `select` (liste).

Qui di seguito un esempio di programma con interfaccia grafica.

```
module "/pliant/graphic/ui/server/api.pli"
module "/pliant/graphic/ui/server/draw.pli"
module "/ross/randomlib.ui"
ui_path "/ross/rand"
  window left
    button "exit"
      url_return
  window main
    title "Random numbers"
    para
      ovar Int qty := 10
      input "Quantity: " qty
      select " 0 - " (ovar Str range )
        option "9" "9"; option "99" "99"; option "999" "999"
    button "compute"
      section_overwrite "result"
        var Str result
        eol
        text string:qty + " Casual numbers from 0 to " + range
        result := (decimals range:len * qty); eol
        for (var Int i) 0 result:len
          if ((i % 10) = 0) (eol)
          text (result i*range:len range:len)+ " "
    section "result"
      void
  window bottom
    button "Info"
      section_overwrite "Info"
        text " Condor Informatique - Turin"
    section "Info"
        void
```

Sorgente 1.1-2 Esempio con interfaccia grafica




```
function primeList -> p
  arg Int p
  var DateTime dt := datetime
  var Array:Int prime
  prime += 149; prime += 197
  prime += 257; prime += 313
  prime += 379; prime += 439
  prime += 499; prime += 691
  prime += 829; prime += 977
  (string:dt 18 1) parse (var Int i)   # secondi
  p := prime i
function decimals p -> ris
  arg Int p; arg Str ris
  var Int resto
  ris := repeat p " "
  var Int a := primeList
  resto := a
  resto := 1000 - (1000 \ a) * a
  for (var Int i) 0 (p - 1)
      resto := resto * 10
      ris:i := string:(resto \ a) 0
      resto := resto - (resto \ a)* a
export decimals
```

Sorgente 1.1-3 funzioni pseudorandom