

Introduzione a PERL ed al protocollo CGI



Dario Menasce
Massimo Mezzadri
Marco Rovere



Comitato per la
transizione alle
Nuove Tecnologie
di Calcolo

Bologna, 5-9 Marzo 2001 - **Corso Specialistico a cura della CNTC**

Introduzione

- Cosa sono gli scripting languages
- Perché ci interessa **PERL** in particolare rispetto ad altri linguaggi di scripting
- Guida di sopravvivenza minimale
- Breve introduzione alla sintassi
- Sviluppo di un semplice esempio
- Creazione di un programma relativamente complesso (*stand-alone*)
- Breve introduzione al protocollo **HTML** e **CGI**
- Interfacciamento del programma di esempio ad un **WEB** browser
- Svolgimento di esercizi proposti
- Suggerimenti per reperire documentazione in rete
- Conclusioni

Cos'è uno scripting language?



Scripting language ---> collante tra le funzionalità di un sistema operativo o, più in generale, tra le funzioni primitive di un ambiente applicativo.



Es: shell script
kuip
Tcl/Tk
JavaScript
.....

Gli shell script sono legati non solo al sottostante sistema operativo (**UNIX, Win98...**) ma addirittura alla particolare shell in uso (**sh, bash, tcsh, ...**) (**problemi di portabilità**)
D'altra parte uno shell script per girare necessita di minime risorse di sistema. Questo é molto importante per un sistemista, in quanto una macchina in grado di funzionare in modo minimale é sicuramente dotata almeno di una shell (generalmente la shell sh), per cui gli script di quella shell potranno, in linea di principio, funzionare anche in assenza di ulteriori risorse, quali un interpreter PERL.

Kuip é lo scripting language nativo di **PAW**: abbastanza sofisticato ma non dotato di capacità di interazione col sistema operativo. Vedremo come é possibile usare PERL (con grande vantaggio e soddisfazione) per fargli generare degli script **Kuip** (i cosiddetti **kumac**-files). E' cioè possibile usare PERL come meta-linguaggio



Es: shell script
kumac
Tcl/Tk
JavaScript
.....

Tcl/Tk é una coppia di linguaggi adatti alla creazione di interfacce grafiche. Tcl é sofisticato, ma dotato di strutture di dati non sufficientemente articolate. Esiste un modulo di PERL, chiamato **PERL/Tk**, che sostituisce Tcl nella coppia di linguaggi: é possibile con esso scrivere sofisticate interfacce utente, come ad esempio un completo browser o un'interfaccia ad un DAQ.

JavaScript é uno scripting language che opera **ESCLUSIVAMENTE** all'interno di un WEB browser (tipicamente Netscape o InternetExplorer). é utilizzato per aggiungere funzionalità dinamiche alle pagine WEB, come ad esempio verificare la validità dei campi di input di un **form** prima che questo venga spedito al server. Questo risparmia al server di verificare la validità dei dati in arrivo da centinaia di clients, demandando ad essi tutta la computazionalità necessaria.



Domanda:

io che sono un fisico e non uno studioso di informatica, quali vantaggi posso trarre dall'uso di **PERL** nella mia attività quotidiana?

E perché linguaggi come il **FORTRAN** o gli **shell scripting** possono non essere ugualmente adeguati allo scopo?

Semplice da imparare

La curva di apprendimento ha una derivata piccola. Si possono facilmente riciclare le conoscenze di **C** e di **shell programming**

Molto sofisticato e flessibile

Pur essendo semplice da imparare, è pur sempre un linguaggio estremamente ricco di attributi e funzionalità

Portabile

Questo è uno dei maggiori punti di forza di PERL: una volta scritto il codice, esso funzionerà senza modifiche sul 99% delle piattaforme esistenti.

Ottimo collante fra processi

PERL è nato per manipolare file e stringhe, ma si è evoluto fino a divenire un ottimo collante fra procedure eterogenee, poiché sa interagire col soggiacente sistema operativo in modo molto efficace.

è estremamente sintetico

La sua sintassi è molto concisa: ciò permette di descrivere in poche istruzioni algoritmi piuttosto complessi. Ciò potenzialmente minimizza la possibilità di introdurre errori involontari nel codice



é interpretato

Pur essendo interpretato é piuttosto veloce in esecuzione. Inoltre ha la caratteristica di avere accesso a run-time al proprio compilatore

La sua velocità non é però comparabile a quella di un programma compilato (inadatto ad applicazioni che siano critiche nei tempi di esecuzione, tipo DAQ o real-time)

Prima di passare ad una descrizione del linguaggio PERL, vediamo se il nostro computer ha una versione installata e recente dell'interprete.



```
> which perl
```

Fornisce il full path-name dell'eseguibile purché sia in uno di quelli specificati nella variabile **PATH**

```
> whereis perl
```


Cerca l'eseguibile, e le eventuali man-pages associate in tutto il tree / (root directory tree)

```
> locate perl
```

Cerca in un apposito database del sistema (solo su **LINUX**) tutte le istanze di files contenenti la stringa **perl** interpretata come **regular-expression**

```
> perl -v
```

Versione 5.004_01



```
This is perl, version 5.004_01
```

```
Copyright 1987-1997, Larry Wall
```

```
Perl may be copied only under the terms of either the Artistic  
Licence or the GNU General Public Licence, which may be found in  
the PERL 5.0 source kit.
```

```
> perl -w source.pl
```

Interpreta il file **source.pl**, stampa eventuali warnings di compilazione, e lo esegue

```
> perl -c source.pl
```

Interpreta il file **source.pl**, stampa eventuali warnings di compilazione, ma **NON** lo esegue



La sintassi di PERL

I tipi di variabili

Gli scalari

```
$Number = 16 ;  
$mass = 1.868 ;  
$myName = "Dario Menasce" ;  
$Address = "Via Celoria" ;  
$FullAdd = "$myName $Address $Number." ;  
  
print("$FullAdd\n") ;
```

Come in C, anche in PERL un'istruzione é terminata dal ;

Una variabile scalare può contenere anche altre entità, ad esempio pointers

```
$Reference = \$Variable ;
```

\$Reference contiene l'indirizzo in memoria della variabile \$Variable . Ad esempio:

```
$Variable = "alfa" ;
```

```
> chmod +x test.pl  
> perl -w test.pl  
Dario Menasce, Via Celoria 16.
```

test.pl

\$myName

\$Address

\$Number

\$Variable → alfa 0x14002a198

\$Reference -> 0x14002a198



La sintassi di PERL

I tipi di variabili

I vettori

```
@VectorA = (1,2,3,4,5,6) ;
```

Il nome collettivo di un vettore é identificato dal simbolo @

Come in C, i vettori in PERL partono dall'indice 0 !!!
Un singolo elemento di un vettore é uno scalare!

```
@VectorB = ("Bob",25,$VectorA[3],(25,26,'String') ) ;
```

```
@VectorC = (@VectorB,$VectorA[0]) ;
```

```
print("@VectorA\n@VectorB\n@VectorC\n" ) ;
```

Vediamo alcune delle operazioni utili sui vettori
che é possibile realizzare con una singola istruzione
(i fisici amano moltissimo i vettori...)

```
Curso Specialistico di ...  
<CNTC> ./test.pl  
1 2 3 4 5 6  
Bob 25 4 25 26 String  
Bob 25 4 25 26 String 1  
<CNTC> █
```



La sintassi di PERL (I)

```
@fred = (1, "ab", $var) ;
```

Inizializziamo un vettore con tre elementi:

```
@barney = @fred ;
```

Ne facciamo una copia

```
$length = @fred ;
```

Calcoliamo il numero di elementi del vettore \$length varrà 3

```
($one) = @fred ;
```

Estraiamo il primo elemento del vettore
(equivalente a \$one = \$fred[0] ossia 1)

```
($a,$b,$c) = @fred ;
```

Estraiamo i singoli elementi del vettore
\$a = 1, \$b = "ab" e \$c = \$var

```
($b,$a) = ($a,$b) ;
```

Scambiamo \$a con \$b

```
$fred[1] = "alpha" ;
```

Riassegna un valore al secondo elemento del vettore

```
$k = ++$fred[0];
```

Incrementa il primo elemento del vettore (\$k diventa 2)

```
$j = $fred[0]++;
```

Assegna a \$j il valore \$fred[0](1) e solo dopo incrementa \$fred[0] di una unità



La sintassi di PERL (II)

```
@fred = (1, "ab", $var) ;
```

```
$new = $fred[1]++;
```

Incrementa il secondo elemento del vettore: poiché è una stringa, incrementa il valore ASCII dell'ultimo carattere della stringa (`$new` diventa `ac`)

```
@fred[1,2] = @fred[2,1];
```

Scambia fra loro il secondo e terzo elemento del vettore (operando su slices del vettore)

```
$fred[2] += 4 ;
```

Somma un 2 a `$var`: equivale a `$fred[2] = $fred[2] + 4;`

```
$last = $fred[-1] ;
```

Restituisce l'ultimo elemento del vettore (`$var`)

```
$lastind = $#fred ;
```

Restituisce l'indice dell' ultimo elemento del vettore (in questo caso `2`)

```
$fred[8] = "$pippo" ;
```

Aumenta la dimensione del vettore a 9 elementi definendo l'ultimo: gli elementi `[4],[5],[6],[7]` avranno valore undef

```
$fred[1] .= "cd" ;
```

Concatena al secondo elemento del vettore la stringa `"cd"` (si ottiene `"abcd"`)



La sintassi di PERL

Sono possibili manipolazioni arbitrariamente complesse sugli elementi di un vettore: come esempio, supponiamo di dover riordinare gli elementi di un vettore secondo un criterio che sia espresso tramite indici numerici immagazzinati in un file (scritti da un altro programma)

```
@fred = ("sette", "otto", "nove") ;
```

Inizializzo un vettore con a elementi

```
@ord = (2, 1, 0) ;
```

Inizializzo un secondo vettore a tre elementi contenente tre indici che rappresentano l'ordine con cui voglio riassemblare il primo vettore (l'esempio ha più senso se lo immaginiamo applicato a qualche decina di migliaia di elementi, nel qual caso li leggeremo da un file opportuno).

```
@back = @fred[ @ord ] ;
```

```
@back = @fred[2, 1, 0] ;
```

Ciò é equivalente a:

Che é a sua volta equivalente a:

```
@back = ($fred[2], $fred[1], $fred[0]) ;
```

```
@back = ("nove", "otto", "sette") ;
```

Abbiamo quindi riordinato un vettore secondo un algoritmo di riallocazione la cui descrizione consiste semplicemente in un vettore di indici la cui origine può essere qualsiasi cosa, un file, un algoritmo, l'output di un sistema di acquisizione...

Vedremo più avanti come sia possibile ottenere lo stesso risultato in modo più semplice



La sintassi di PERL

Gli operatori di *stacking*:

push, *pop*, *shift* and *unshift*



```
@mylist = (1, "two", "three") ;  
$newvalue = 4 ;  
push @mylist, $newvalue ;
```

```
print( "mylist = @mylist\n" ) ;  
  
mylist = 1 two three 4
```

Ai vettori é associato un ricco assortimento di operatori capaci di agire su di essi in modo sintetico. Vediamone alcuni:

Stack

push estende la dimensione del vettore e aggiunge l'ultimo elemento in coda

\$mylist[0]	1
\$mylist[1]	two
\$mylist[2]	three
\$mylist[3]	4



La sintassi di PERL

Gli operatori di *stacking*:

push, *pop*, *shift* and *unshift*



```
$lastadded = pop( @mylist ) ;
```

```
print( "Last added = $lastadded\n" ) ;
```

```
Last added = 4
```

`$mylist[0]`

1

`$mylist[1]`

two

`$mylist[2]`

three

pop rimuove l'ultimo elemento dello stack
(la gestione degli indici é a carico di *pop*)



La sintassi di PERL

Gli operatori di *stacking*:

push, *pop*, *shift* and *unshift*



```
$mylist = (1,"two","three") ;  
unshift( @mylist, "zero" ) ;
```

```
print( "mylist = @mylist\n" ) ;  
  
mylist = zero 1 two three
```

unshift agginunge un elemento al primo posto nel vettore: poiché rinumera gli elementi, può essere inefficiente per vettori molto grandi.



`$mylist[0]`

zero

`$mylist[1]`

1

`$mylist[2]`

two

`$mylist[3]`

three



La sintassi di PERL

Gli operatori di *stacking*:

push, *pop*, *shift* and *unshift*



```
$out = shift( @mylist) ;
```

```
print( "Taken out = $out\n" ) ;
```

```
Taken out = zero
```

shift rimuove il primo elemento dello stack: di nuovo, viene rinumerato tutto il vettore.



`$mylist[0]`

zero

`$mylist[1]`

1

`$mylist[2]`

two

`$mylist[3]`

three



La sintassi di PERL

Gli operatori di ordinamento e sorting:

```
@dritto = (1, 2, 3, 4) ;  
@rovescio = reverse( @dritto ) ;
```

```
@sparso = ("gamma", "beta", "alfa" ) ;  
@ordinato = sort (@sparso) ;
```

```
print("@rovescio\n") ;
```

```
4 3 2 1
```

```
print("@ordinato\n") ;
```

```
alfa beta gamma
```

```
@list = <STDIN> ;
```

Un programma fatto da questa sola istruzione riempie ogni elemento del vettore **list** con una riga digitata in input. Il procedimento termina quando viene digitato un **EOF** (^D)



La sintassi di PERL

I tipi di variabili

Le hash tables

Una hash table é una collezione di scalari raggruppati sotto un unico nome, la cui posizione é identificata, invece che da un numero intero positivo, come accade nel caso dei vettori, da una **arbitraria stringa di caratteri**.

```
$sede{"Dario Menasce"} = "Milano" ;  
$sede{"$utente"}      = "Torino" ;
```

Questa sintassi é equivalente a:

```
%sede = (  
    "Dario Menasce" => "Milano"  
    "$utente"      => "Torino"  
);
```

Ed é anche equivalente a:

```
%sede = ("Dario Menasce", "Milano", $utente, "Torino") ;
```

```
@lista = %sede ;
```

Una hash può essere trasformata in una lista...

```
%nuovasede = @lista ;
```

... e viceversa



La sintassi di PERL

I tipi di variabili

Le hash tables

Le hash sono le strutture dati native di PERL dotate della maggior versatilità per la costruzione di strutture più complesse, come hash di hash ecc...

```
$FirstName = "Dario" ;  
$LastName  = "Menasce" ;  
$City      = "Milano" ;  
$Employee  = "$FirstName $LastName" ;  
$Street{"Dario Menasce"} = "Via Celoria 16" ;
```

```
$Address{"$FirstName-$LastName"} = "$Street{$Employee}" . " $City" ;
```

```
print("Address of $FirstName-$LastName: $Address{$FirstName-$LastName} \n" ) ;
```

```
Address of Dario Menasce: via Celoria 16 Milano
```



La sintassi di PERL

I tipi di variabili

Gli operatori sulle hash tables

L'operatore `each`

```
$Hash{"alpha"} = "a" ;  
$Hash{"beta"}  = "b" ;  
$Hash{"rho"}   = "r" ;  
$Hash{"pi"}    = "p" ;
```

L'operatore `each` restituisce, in un contesto vettoriale, tutte le coppie chiave/valore definite per la hash in questione

```
while (($greek,$latin) = each(%Hash)) {  
    print ("La lettera greca $greek corrisponde alla latina $latin \n") ;  
}
```

```
La lettera greca pi corrisponde alla latina p  
La lettera greca rho corrisponde alla latina r  
La lettera greca beta corrisponde alla latina b  
La lettera greca alpha corrisponde alla latina a
```



La sintassi di PERL

I tipi di variabili

```
$Hash{"alpha"} = "a" ;  
$Hash{"beta"}  = "b" ;  
$Hash{"rho"}   = "r" ;  
$Hash{"pi"}    = "p" ;  
  
@List = keys ( %Hash ) ;  
print ("@List\n") ;
```

pi rho beta alpha

```
@List = sort keys ( %Hash ) ;  
print ("@List\n") ;
```

alpha beta pi rho

```
@List = sort values ( %Hash ) ;  
print ("@List\n") ;
```

a b p r

Gli operatori sulle hash tables

L'operatore keys

Queste istruzioni sono assolutamente equivalenti al comando `printenv`

```
foreach $key ( sort keys %ENV ) {  
    print ("$key=$ENV{$key}\n") ;  
}
```

L'ordine non é quello di inserimento, bensì quello determinato dall'algoritmo di hashing. Se vogliamo una lista ordinata alfabeticamente useremo l'operatore `sort`

L'operatore `values` agisce sui valori della hash in modo analogo:



La sintassi di PERL

I tipi di variabili

Gli operatori sulle hash tables

Esistono innumerevoli modi per assegnare elementi ad una hash

```
$record{"Fred"} = 205 ;  
$record{"Barney"} = 195 ;  
$record{"Wilma"} = 30 ;
```

Ad ogni personaggio associamo il suo record personale a bowling...

Queste due forme di assegnazione sono completamente equivalenti

```
@record{"Fred", "Barney", "Wilma"} = (205, 195, 30) ;
```

parentesi graffe!!

tonde!!

Un altro modo ancora...

```
@Vincitori = ("Fred", "Barney", "Wilma") ;  
@Valori = (205, 195, 30) ;  
@record{@Vincitori} = @Valori ;
```

Indipendentemente da come @record viene assegnato...

```
while (($persona, $valore) = each %record) {  
    print("Il record di $persona é $valore\n" ) ;  
}
```

```
Il record di Barney é 195  
Il record di Fred é 205  
Il record di Wilma é 30
```



La sintassi di PERL

Le strutture di controllo

Analoghe a quelle di molti linguaggi (tipo C)

Qualsiasi espressione che, una volta valutata, dia un valore diverso da zero

```
if (condition1) {  
    block1 ;  
}  
elseif (condition2) {  
    block2 ;  
}  
else {  
    block3 ;  
}
```

```
while (condition) {  
    block ;  
}
```

```
until (condition) {  
    block ;  
}
```

```
do {  
    block ;  
} while (condition);
```

```
do {  
    block ;  
} until (condition);
```

In questo caso, prima viene eseguito il blocco di istruzioni block, poi viene valutata la condizione condition per eventualmente reiterare il blocco

```
unless (condition) {  
    block ;  
}
```

```
for(init;test;incr){  
    block ;  
}
```

```
foreach $i (@list) {  
    block ;  
}
```



La sintassi di PERL

I sottoprogrammi

Sintassi simile a quella di molti linguaggi (tipo C)

```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
'menasce/CorsoSpecialistico/temp.pl line 11, col 1, 133 bytes
#!/usr/local/bin/perl

Stampa() ;

#-----
sub Stampa {
    print("Testo\n") ;
}
```

Chiamata al sottoprogramma

Implementazione del sottoprogramma

```
menasce@almifome
<CNTC> ./temp.pl
Testo
<CNTC> █
```



La sintassi di PERL

I sottoprogrammi

Sintassi simile a quella di molti linguaggi (tipo C)

```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
menasce/CorsoSpecialistico/temp.pl line 13, col 1, 194 bytes
#!/usr/local/bin/perl

Stampa("Primo testo");
Stampa("Secondo testo");

#-----
sub Stampa {
  my $What = shift;
  print("$What\n");
}
```

```
menasce@almifome
<CNTC> ./temp.pl
Primo testo
Secondo testo
<CNTC> █
```

Stampa("Primo testo") ;

@_

L'argomento formale del sottoprogramma corrisponde al vettore implicito di PERL @_ (in questo esempio il vettore @_ conterrà il solo elemento stringa "Primo testo").

Le variabili in PERL, se non specificato altrimenti, sono sempre **globali** (sono visibili da ogni sottoprogramma). La keyword **my** specifica che la variabile indicata (**\$What** nell'esempio) é definita unicamente nel sottoprogramma Stampa (si dice che é locale al gruppo di parentesi graffe che la racchiudono, **body**)

```
Sub Stampa {
  print("@_[0]\n") ;
}
```



La sintassi di PERL

I sottoprogrammi possono ricevere un input (argomenti formali) e restituire un output (che può essere una qualsiasi struttura)

I sottoprogrammi

Definiamo un vettore

```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
menasce/CorsoSpecialistico/temp.pl 251 bytes
#!/usr/local/bin/perl

@Vector = (1, 2, 3, 4) ;

$Result = &Somma(@Vector) ;
print("$Result\n" ) ; ;

# -----

sub Somma {
  my $Tot = 0;
  foreach $i (@_) {
    $Tot += $i ;
  }
  return $Tot ;
}
```

Invochiamo un sottoprogramma che calcola la somma degli elementi di un vettore fornito come argomento formale.

Definiamo ed inizializziamo una variabile locale che conterrà il risultato della somma.

Preleviamo un elemento alla volta dal vettore implicito di ingresso, @_

Sommiamo i valori prelevati da @_

Il sottoprogramma restituisce uno scalare, **\$Tot**, che contiene la somma calcolata. Il valore restituito viene messo nella variabile **\$Result** che viene infine stampata sullo STANDARD OUTPUT.



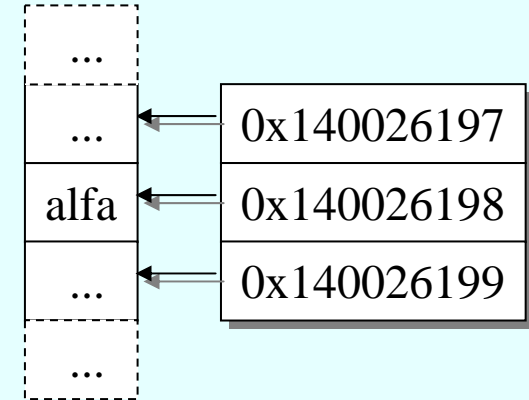
La sintassi di PERL

Le references (o pointers)

Il concetto di puntatore ad una variabile é fondamentale: esso permette la definizione e manipolazione di complesse strutture di dati, pur mantenendo una sintassi leggera.

Consideriamo la seguente istruzione: `$a = "alfa" ;` il valore `alfa` viene associato alla variabile `$a` tramite un indirizzo di memoria nel quale viene immagazzinato il valore.

```
$a      = "alfa" ;      named scalar ref
$ref_a = \ $a      ;
print("a = $a, ref_a = $ref_a\n" ) ;
```



```
<CNTC> ./pointers.pl
a = alfa, ref_a = SCALAR(0x140026198)
<CNTC> █
```

type **address**

In PERL una referenza ha un attributo di *tipo*: una referenza ad uno scalare é trattata in modo differente da una referenza ad un vettore o ad una hash. Questo ha importanti conseguenze che vedremo tra poco: notiamo che tentativi da parte nostra di utilizzare referenze di tipo in scalare in contesti vettoriali generano errore da parte dell'interprete.

In PERL esistono referenze a variabili dotate di nome come pure referenze a strutture cosiddette *anonime*.



La sintassi di PERL

Le references (o pointers)

Nome dell'array

array

Operatore di referenziazione

array

```
@a = ("alfa", "beta");
$ref_a = \@a;
print("a = @a, ref_a = $ref_a \n" );
foreach $i (@$ref_a) {
  print("i = $i \n" );
}
```

named array ref

```
$ref_a = [ "alfa", "beta" ];
print("ref_a = $ref_a \n" );
foreach $i (@$ref_a) {
  print("i = $i \n" );
}
```

anonymous array ref

```
Curso Specialistico PERL
<CNTC> ./pointers.pl
a = alfa beta, ref_a = ARRAY(0x1400261a8)
i = alfa
i = beta
<CNTC> █
```

type

```
Curso Specialistico PERL
<CNTC> ./pointers.pl
ref_a = ARRAY(0x140019868)
i = alfa
i = beta
<CNTC> █
```

A cosa servono i puntatori?



La sintassi di PERL

Le references (o pointers)

I puntatori sono utili in quanto permettono di maneggiare complesse strutture di dati usando unicamente l'indirizzo di partenza della struttura in memoria.

Esistono moltissime applicazioni di questo concetto: vediamone alcune mediante esempi

Consideriamo il caso di una **hash**. Abbiamo visto che una **hash** associa un valore (uno scalare) ad una chiave (sempre uno scalare). L'associazione è però univoca, nel senso che ad una particolare chiave si può associare uno ed un solo un valore:

```
$hash{"temperatura"} = 25 ;  
$hash{"temperatura"} = 78 ;
```

Questa seconda istruzione **ridefinisce** l'associazione

L'uso delle referenze permette di circoscrivere questa limitazione. Vediamo come:

```
$hash{"temperatura"} = [25,78,34,-12] ;
```

Questa istruzione associa all'elemento di hash individuato dalla chiave **temperatura** la **referenza** al vettore (25,78,34,-12)

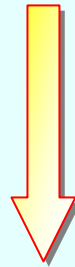
Notare l'uso delle parentesi quadre: esse stanno ad indicare che l'oggetto definito a destra del segno di uguaglianza non è un vettore (avremmo usato le parentesi tonde), bensì una **referenza al vettore** (vettore anonimo, in quanto non dotato di nome)



La sintassi di PERL

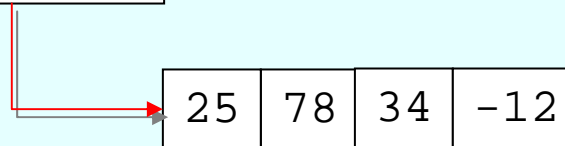
Le references (o pointers)

```
$hash{"temperatura"} = [25,78,34,-12] ;
```



L'operatore parentesi quadre crea una referenza al vettore che racchiude al suo interno. Questa referenza viene associata all'elemento di hash indicato

```
$hash{"temperatura"} 0x140019868
```



Abbiamo quindi utilizzato la struttura di hash in modo sintatticamente corretto: abbiamo associato ad una chiave uno scalare (**temperatura** \Leftrightarrow **0x140019868**). Poiché però lo scalare é in realtà un puntatore ad un vettore, in definitiva abbiamo associato ad una chiave un insieme strutturato di elementi.

Vediamo un programmino di esempio completo che faccia uso della tecnica descritta:



La sintassi di PERL

Le references (o pointers)

```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl line 14, col 0, 223 bytes
#!/usr/local/bin/perl

$hash{"temperatura"} = [25,78,34,-12] ;

print <<End_OfText ;

Referenza associata alla chiave "temperatura": $hash{"temperatura"}
Valori corrispondenti: @{$hash{"temperatura"}}

End_OfText
```

Il costrutto `@{referenza}` ci permette di accedere direttamente al vettore referenziato per recuperarne i valori

```
menasce@almifome
<CNTC> ./temp.pl

Referenza associata alla chiave temperatura: ARRAY(0x140019868)

Valori corrispondenti: 25 78 34 -12

<CNTC> █
```

Con la sola chiave, *temperatura*, abbiamo memorizzato un intero vettore di valori (25, 78, 34, -12)

Abbiamo descritto uno solo fra gli innumerevoli modi in cui il costrutto di puntatore o referenza risulta non solo di estrema utilità ma spesso é addirittura il solo modo per risolvere certi problemi.



La sintassi di PERL

Operazioni di I/O

Il concetto di *file-handle*

Come si gestisce un file in PERL? Come si fa a leggerne il contenuto? Come si fa a scrivere o ad aggiornare un file? é molto semplice, più che in ogni altro linguaggio. A questo scopo é fondamentale il concetto di *file-handle*, vagamente analogo al concetto di *I/O unit* in **FORTRAN**, ma dotato di ben più generali e potenti proprietà.

```
open (IN, "myfile.dat") ;  
while ( $_ = <IN> ) {  
    print ("$_\n") ;  
}  
close(IN) ;
```

Un *filehandle* é il nome dato ad una connessione fra il corrente processo di PERL ed il mondo esterno: é quindi un canale di I/O con il sottostante sistema operativo, analogo alla **UNIT** del **FORTRAN**

La sintassi estremamente concisa di PERL permette di omettere in molti casi il nome di una variabile specifica: il *\$_* é una variabile implicita di PERL

Questo script si comporta in modo identico al comando UNIX
`cat myfile.dat`



La sintassi di PERL

Operazioni di I/O

Il concetto di file-handle (cont..)

```
open ( LOG, ">myfile.dat" ) || die "Cannot open file myfile.dat" ;  
print LOG ("Process $proc has completed \n" );  
close( LOG ) ;
```

Ridirige ogni output a **LOG** verso il file **myfile.dat**, analogamente a quanto avviene in UNIX con `> o >>`

Se la condizione alla destra del simbolo di or, `||`, è falsa, ossia se il file non può essere aperto, il programma muore producendo la frase indicata a destra

Esiste una ricca serie di qualificatori capaci di restituire informazioni circa un file; il loro uso è molto semplice:

```
$file = "myfile.dat" ;  
if (-e "$file") {  
    print("File $file already exists!\n") ;  
}
```



La sintassi di PERL

Operazioni di I/O

Come faccio a mettere in un vettore la lista dei files di una directory che iniziano con la stringa **host** seguito da qualsiasi carattere?

Con una sola istruzione otteniamo i seguenti risultati:

- con gli operatori **<** e **>** abbiamo aperto un canale di **I/O**
- con la wild-card ***** specifichiamo un intero range di nomi
- l'output dell'operatore **< >** consiste in una lista di oggetti e viene quindi valutato in un contesto vettoriale: di conseguenza ogni oggetto viene inserito in un elemento del vettore **list**

Il concetto di file-globbing

```
@list = </etc/host* > ;
```

```
foreach $file (@list) {  
    print ("$file") ;  
}
```

In modo ancora più conciso:

```
foreach (</etc/host*>) {  
    print ;  
}
```

Se il nome del file é costituito da una variabile, non potremo usare l'operatore bra-ket (<>) ma dovremo usare l'operatore **glob**, dotato delle stesse funzioni ma più generale.


```
$name = "host" ;  
$filename = "/etc/${name}*" ;  
@list = glob $filename ;
```



La sintassi di PERL

Operatori di interazione col `filesystem` locale

Un altro punto di forza di PERL é la sua portabilità : questa é realizzata creando un **abstraction layer** tra la funzionalità e l' implementazione di un comando diretto al sistema operativo. Vediamo alcuni esempi utili:

`unlink ("myfile.dat") ;`  `rm myfile.dat`
Equivale, in **UNIX**, a:

L'**abstraction layer** consiste quindi nel disaccoppiare un comando dalla sua specifica sintassi in un particolare sistema operativo. Su **VMS** `unlink` corrisponde a **delete**, e via scorrendo

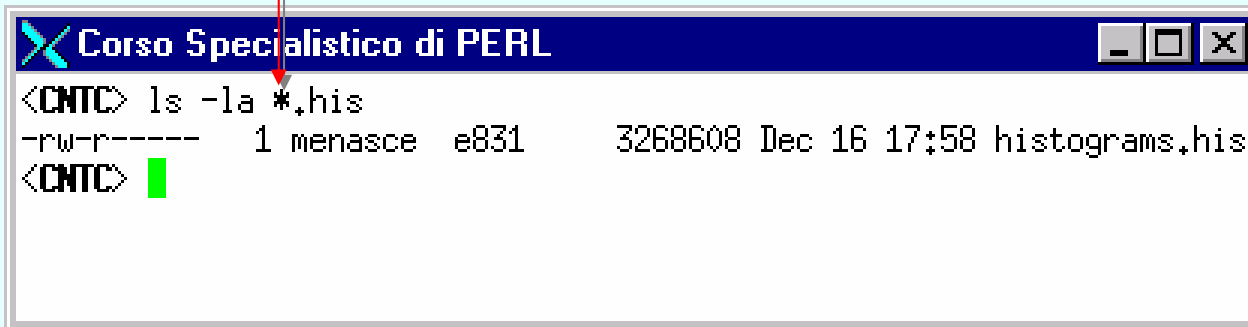
<code>rename ("myfile.dat myfile.old") ;</code>	<code>mv myfile.dat myfile.old</code>
<code>link ("fred barney") ;</code>	<code>ln fred barney</code>
<code>symlink ("fred barney") ;</code>	<code>ln -s fred barney</code>
<code>mkdir ("newdir", 0755) ;</code>	<code>mkdir newdir; chmod 755 newdir</code>

L'elenco non finisce qui: l'uso di questi comandi rende un programma estremamente portabile tra piattaforme, e se ne consiglia quindi vivamente l'uso.



Le regular expressions costituiscono un importantissimo strumento di lavoro: esse sono, in sintesi, un modo estremamente conciso e potente per descrivere dei pattern di caratteri mediante l'uso di una particolare sintassi.

Un esempio triviale di regular expression e' la cosiddetta wild-card di UNIX.



```
<CNTC> ls -la *.his
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
<CNTC> █
```

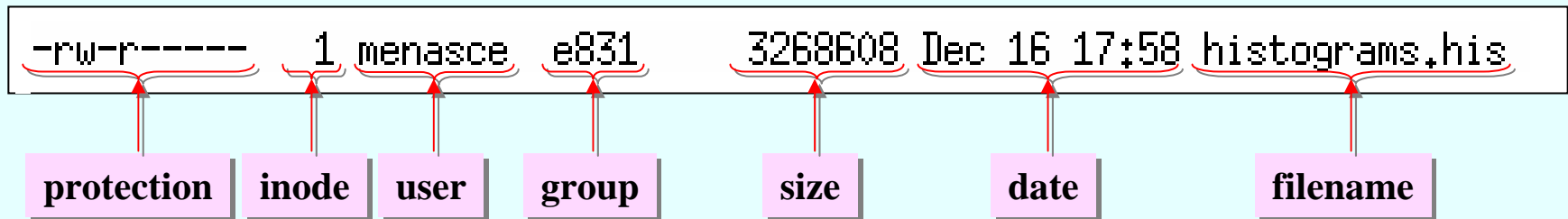
In questo contesto, il simbolo **asterisco** ha il seguente significato: **qualunque carattere seguito dai caratteri ., h, i ed s**. Con un solo simbolo possiamo dunque descrivere un pattern di grande generalità (consideriamo il caso, ad esempio di `ls -la *gra*.h*s`).

In qualche modo possiamo dire che una regular expression é un'estensione logica di questo concetto (affermazione impropria ma esemplificatrice)



La sintassi delle regular expression é senza dubbio criptica, a fronte però di una grandissima generalità e potenzialità di sintesi. Chiariamo la cosa con esempi:

Supponiamo di voler verificare se una certa variabile contiene una stringa di caratteri che sia compatibile con la specifica di un file ottenuta con il comando `ls`, ossia che la stringa sia della forma:



Notiamo come la stringa contenga dei caratteri blank come separatori

Una regular expression che descriva sinteticamente questo pattern sarà del tipo:

`pattern spazi pattern spazi pattern` dove al posto di pattern ci sarà di volta in volta un elemento della sintassi delle reg-exp che descrive il particolare pattern cercato (ad esempio `17:58` sarà rappresentato dal pattern `digits:digits`)



Vediamo ora in dettaglio cosa prescrive la sintassi delle regexp per questo caso:

```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

`\s` é un meta-character che rappresenta il carattere blank (spazio bianco)

`\s+` indica che ci aspettiamo **almeno un** carattere bianco

Un primo pezzo della regular expression che descrive il pattern qui sopra é pronto

Un certo numero di spazi

`\s+`

`\s+`

`\s+`

`\s+`

`\s+`

`\s+`

`\s+`

`\s+`



```
-rw-r-----
```

```
1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

La riga inizia con la *protection-mask*, costituita da una serie di caratteri sia alfabetici che, possibilmente, dal segno -

10 caratteri

Il *meta-character* che descrive un qualsivoglia carattere é il punto (`.`). Poiché ce ne sono esattamente 10, specifichiamo la quantità mediante `{10}`

La regular expression sta crescendo...

```
.{10} \s+ \s+ \s+ \s+ \s+ \s+ \s+
```




```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

Lo *inode*, che segue la protection-mask e precede lo owner del file, é rappresentato mediante un numero intero a più cifre.

Il *meta-character* che descrive un digit é `\d`. Come sempre, poiché ve ne possono essere più di uno, scriveremo `\d+`

`.{10}` `\s+` `\d+` `\s+` `\s+` `\s+` `\s+` `\s+` `\s+` `\s+`



```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

Lo *owner*, invece, é costituito da un insieme di caratteri alfanumerici

Il *meta-character* che descrive un carattere alfanumerico é `\w`.
Come sempre, poiché ve ne possono essere più di uno, scriveremo `\w+`

`.{10}` `\s+` `\d+` `\s+` `\w+` `\s+` `\s+` `\s+` `\s+` `\s+` `\s+`



```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

Stessa cosa per il *group*,
un insieme di
caratteri alfanumerici

```
.{10} \s+ \d+ \s+ \w+ \s+ \w+ \s+ \s+ \s+ \s+
```



```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

A questo punto il gioco é semplice: i *pattern* che rimangono sono dello stesso tipo di quelli già considerati, *digits*, e/o caratteri.

`.{10}` `\s+` `\d+` `\s+` `\w+` `\s+` `\w+` `\s+` `\s+` `\s+` `\s+` `\s+`



La sintassi di PERL

Le regular-expressions

```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

Possiamo specificare che fra le due quantità numeriche ci aspettiamo il simbolo “:”, *verbatim*

```
.{10} \s+ \d+ \s+ \w+ \s+ \w+ \s+ \d+ \s+ \w+ \s+ \d+ \s+ \d+:\d+ \s+
```



```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.hist
```

Qualsiasi altro carattere alfanumerico rimane, fa certamente parte del nome del file, per cui specificheremo nella nostra regexp il *meta-character* “.” per un numero arbitrario di occorrenze.

La nostra regexp é ora completa. Usiamola in un programma reale:

```
.{10} \s+ \d+ \s+ \w+ \s+ \w+ \s+ \d+ \s+ \w+ \s+ \d+ \s+ \d+:\d+ \s+ .+
```



```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

temp.pl

File Edit Search Preferences Shell Macro Windows Help

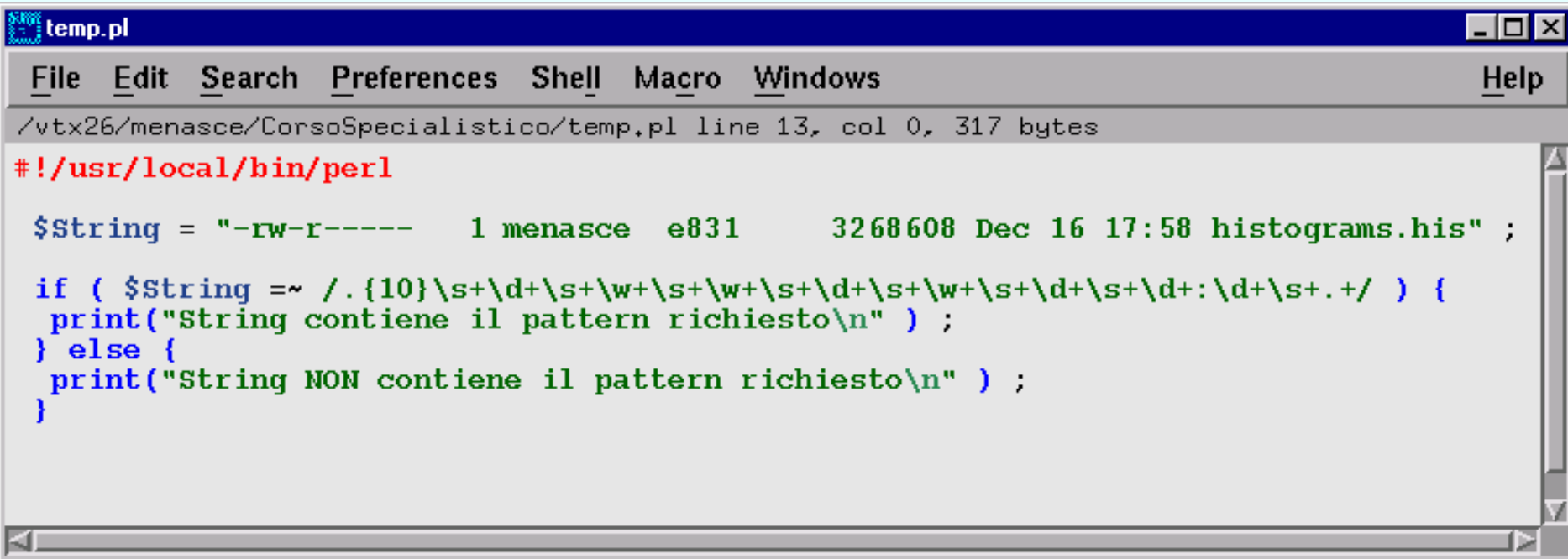
/vtx26/menasce/CorsoSpecialistico/temp.pl line 13, col 0, 321 bytes

```
#!/usr/local/bin/perl
```

```
$String = "-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his" ;
```

Definiamo una variabile con la stringa di cui vogliamo verificare l'aderenza alla sintassi del comando **ls -la**





```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl line 13, col 0, 317 bytes
#!/usr/local/bin/perl

$string = "-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his" ;

if ( $string =~ /.{10}\s+\d+\s+\w+\s+\w+\s+\d+\s+\w+\s+\d+\s+\d+:\d+\s+.+/ ) {
    print("String contiene il pattern richiesto\n" ) ;
} else {
    print("String NON contiene il pattern richiesto\n" ) ;
}
```

```
.{10}\s+\d+\s+\w+\s+\w+\s+\d+\s+\w+\s+\d+\s+\d+:\d+\s+.+
```




```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

Supponiamo ora di voler estrarre dalla stringa in questione il nome del file e la sua dimensione.

Se nella regular expression circondiamo di parentesi tonde la parte che descrive il pattern che vogliamo estrarre, PERL riempirà, per nostro conto, delle variabili implicite con il contenuto dell'eventuale *match* trovato. Con la seguente sintassi:

```
.{10}\s+\d+\s+\w+\s+\w+\s+(\d+)\s+\w+\s+\d+\s+\d+:\d+\s+(.+)
```

Possiamo ora memorizzare le variabili \$1 e \$2 in altre due, dotate di nome:

\$1

\$2



```
-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his
```

Il nostro programma diviene a questo punto:

```
.{10}\s+\d+\s+\w+\s+\w+\s+(\d+)\s+\w+\s+\d+\s+\d+:\d+\s+(.+)
```

Possiamo ora memorizzare le variabili \$1 e \$2 in altre due, dotate di nome:

```
$size = $1
```

```
$1
```

```
$file = $2
```

```
$2
```



```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl line 15, col 0, 349 bytes
#!/usr/local/bin/perl

$string = "-rw-r----- 1 menasce e831 3268608 Dec 16 17:58 histograms.his" ;

if ( $string =~ /.{10}\s+\d+\s+\w+\s+\w+\s+(\d+)\s+\w+\s+\d+\s+\d+:\d+\s+(.+)/ ) {
    $size = $1 ;
    $file = $2 ;
    print("Il file $file contiene $size bytes\n" ) ;
} else {
    print("String NON contiene il pattern richiesto\n" ) ;
}
```

```
menasce@almifo1e
<CNTC> ./temp.pl
Il file histograms.his contiene 3268608 bytes
<CNTC> █
```



```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl 356 bytes
#!/usr/local/bin/perl

$/ = ' ' ;

while(<DATA>) {
  while ( m{
    \b
    (\s+)
    \b
    (
      \s+
      \1
      \b
    ) +
  }xig
  )
  {
    print("La parola $1 compare due volte nel testo\n" ) ;
  }
}

__DATA__
Questo esempio serve per una prova
prova della esistenza di parole
duplicate varie varie volte in un testo
```

L'esempio discusso finora é assolutamente rudimentale: unicamente allo scopo di mostrare le potenzialità di una regular expression, vediamo un esempio nel quale, una sola stringa di meta-character descrive una regexp capace di individuare parole ripetute in un testo:



```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl 356 bytes
#!/usr/local/bin/perl

DATA
Questo esempio serve per una prova
prova della esistenza di parole
duplicare varie varie volte in un testo
```

Questo costrutto prepara una sorta di file interno, dal quale PERL può leggere dei dati. Utile per piccole prove di programmazione.



```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl 356 bytes
#!/usr/local/bin/perl

$/ = ' ';

DATA
Questo esempio serve per una prova
prova della esistenza di parole
duplicate varie varie volte in un testo
```

Affinché sia possibile **immagazzinare** tutte le linee del testo qui sotto indicato **in una sola stringa**, usiamo la variabile implicita di PERL \$/, che contiene il carattere usato da PERL per rappresentare il terminatore di record (generalmente il <CR>). Ridefiniamolo al valore nullo (' ') in modo che il testo venga interpretato come privo di terminatori di record.



```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl 356 bytes
#!/usr/local/bin/perl
$/ = ' ';
while(<DATA>) {
}
__DATA__
Questo esempio serve per una prova
prova della esistenza di parole
duplicie varie varie volte in un testo
```

Predisponiamo un ciclo di lettura dei dati dal canale interno **<DATA>**



```
temp.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/temp.pl 356 bytes
#!/usr/local/bin/perl

$/ = ' ' ;

while(<DATA>) {
  while ( m{
    \b
    (\s+)
    \b
    (
      \s+
      \1
      \b
    ) +
  }xig
  )
  {
    print("La parola $1 compare due volte nel testo\n" ) ;
  }
}

__DATA__
Questo esempio serve per una prova
prova della esistenza di parole
duplicate varie varie volte in un testo
```

Non entriamo nel dettaglio della descrizione della regexp: questo esempio serve unicamente a dare un'idea della grande capacità di sintesi della sintassi delle regexp nel descrivere un particolare pattern (in questo caso la possibile esistenza di parole ripetute anche su righe diverse)



Controllo di processi

È frequente la necessità di interagire con il sistema operativo dall'interno di un processo, sia per ricevere informazioni che per far partire e gestire processi esterni. Vediamo quali strumenti offre PERL per risolvere questo problema assieme a qualche utile esempio.

In quasi tutti i linguaggi di programmazione è possibile fornire dei comandi al sottostante sistema operativo: ciò è utile per diversi motivi:

Quando occorre eseguire un'operazione per la quale esiste un opportuno comando di sistema che non si vuole reinventare, come ad esempio il comando di stampa di un file, **lpr**, o il comando di cancellazione file



Controllo di processi

Il metodo più semplice é l'uso della funzione **system**:

```
system(" ls -la *.txt " ) ;
```

comando al sistema operativo

In seguito all' chiamata a **system**, PERL procede alla creazione (**forking**) di un sottoprocesso, detto processo figlio (**child process**), nell'ambito del quale viene eseguito il comando specificato (il comando viene passato alla shell **sh**).

L'eventuale output prodotto dal comando viene passato allo **STANDARD OUTPUT** del processo padre (PERL), assieme allo **STANDARD ERROR** (si dice che il figlio eredita **STDIN**, **STDOUT** e **STDERR** dal processo padre): generalmente questi canali di comunicazione corrispondono allo schermo del terminale.

```
system("montecarlo.exe &") ;
```

In questo caso viene fatto partire il processo **montecarlo.exe** ma non si aspetta che esso abbia termine per far tornare il controllo al processo padre. L'uso é problematico nel caso il figlio generi dell'output , poiché questo verrà mescolato in modo asincrono con quello del padre.

System é un operatore molto utile, ma l'output generato dal comando eseguito dal sistema operativo, una volta mandato allo **STDOUT**, é perso, non possiamo usarlo all'interno dello script PERL che lo ha lanciato!!!.



Controllo di processi

L'operatore **backticks** (```): un primo modo per catturare l'output di un processo di sistema

```
$now = "Current date is " . `date` ;
```

```
print("$now\n") ;
```

```
Current date is Fri Nov 6 15:00:12 MET 1998
```

Backticks...

Dalla shell, il comando **date** produce il seguente output:

Fri Nov 6 15:00:12 MET 1998

In questo caso, **date** produce una sola linea di output, e quindi i backticks vengono valutati in un contesto scalare ed il risultato posto in una variabile scalare.

```
@ListOfConnectedPeople = `who` ;
```

Il comando **who** genera un output con diverse linee, una per ogni utente: l'output viene quindi valutato in un contesto vettoriale, ed ogni linea verrà memorizzata in un elemento del vettore **@ListOfConnectedPeople**



Controllo di processi

Un modo ancora più flessibile é dato dall'operatore **open** associato ad una **pipe**

```
open( CHI , "who | " ) ;  
open( OUT , " | lpr -Pps_hplit3" ) ;  
while ( <CHI> ) {  
    print OUT ;  
}  
close( CHI ) ;  
close( OUT ) ;
```

Apriamo un altro canale di I/O sul quale manderemo il comando di stampa...

Occorre terminare il comando con il simbolo di **pipe** !!!

Il nome di un file-handle: esso conterrà il canale di I/O associato all'output prodotto dal comando **who**. Occorre specificare che **who** deve ridirigere l'output ad una pipe, ossia allo script corrente.

E se volessimo mandare l'output, eventualmente modificato, direttamente ad una stampante laser? Facile...

Il comando di **print OUT** spedisce ciò che **who** ha prodotto alla pipe **OUT**, ossia al comando **lpr** e quindi alla stampante!!

Qui c'è un semplice **print**: in realtà possiamo manipolare ogni riga di output di **who** agendo sulla variabile **\$_**.

Per mandare degli argomenti ad una pipe, occorre far precedere il comando per il sistema operativo dal simbolo di pipe!!!



La capacità di PERL di interagire con il sistema operativo, nel modo visto dagli esempi precedenti é uno dei motivi che hanno reso PERL così popolare

Consideriamo il seguente esempio: vogliamo scrivere un programma che accetti in input un'arbitraria serie di istruzioni e le esegua.

Una caratteristica interessante di PERL, é la capacità di eseguire una stringa di caratteri, interpretandola come un blocco di istruzioni. La stringa può essere composta leggendo dati dall'esterno del programma: diviene allora possibile scrivere oggetti di estrema generalità come pocket-calculators e simili. Ciò é possibile grazie al fatto che il linguaggio PERL ha accesso al proprio interpreter, al quale può fornire ulteriori parti di programma composte a run-time, dopo che una prima parte del programma é già stata compilata ed eseguita...



Stampa il prompt > sul terminale

Ciò che viene dato in input dal terminale viene **valutato** interpretandolo come espressione in PERL ed il risultato della medesima viene posto nella variabile **\$result** che può essere stampata

```
while(1) {  
  print("> ") ;  
  $result = eval <STDIN> ;  
  print("$result\n") ;  
}
```

Poiché 1 è una costante e vale sempre 1, questo loop è infinito: lo interromperò all'occorrenza con **^C**

La variabile **\$result** viene infine stampata sullo **STDOUT**

Eseguiamo ora il programma dando il seguente input:

```
> $px=12.34;$py=11.34;$pz=123.12; $M=1.869;$p=sqrt($px**2+$py**2+$pz**2);$E=sqrt($M**2+$p**2);  
124.269460290934
```

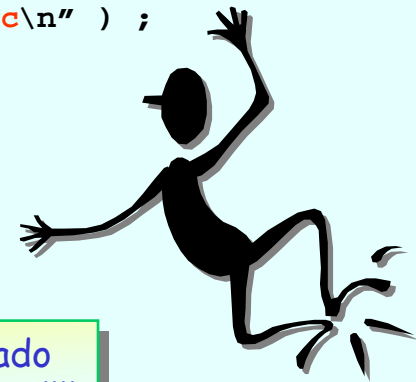
Torna il prompt: **qualunque** altro tipo di calcolo diamo in pasto al programma esso sarà in grado di eseguirlo e darci il risultato!!!!

RETURN

```
> $a=3; $ b=4; $c = sqrt($a**2 + $b**2 ) ;  
5
```

```
> $a=3; $ b=4; $c = sqrt($a**2 + $b**2 ) ; print(" sqrt($a**2 + $b**2 ) = $c\n" ) ;  
sqrt(3**2 + 4**2) = 5
```

La funzione **print** di PERL restituisce 1, valore che viene mandato allo **STDOUT** subito dopo la stringa che print doveva stampare...



Questo semplicissimo programma non solo funge da pocket-calculator, ma è in grado di eseguire qualsiasi parte di un un programma in PERL che gli venga fornito in input!!!!



Un altro importante utilizzo di eval riguarda il cosiddetto **error-trapping** (detto anche **exception-handling**)

```
$a = 14 ;  
print("Enter denominator: " ) ;  
$b = <STDIN> ;  
$r = eval { $c = $a / $b } ;  
if ($@ =~ /Illegal/) {  
    print("Fatal!... $@") ;  
} else {  
    print("Result: $r\n") ;  
}
```

Blocco di istruzioni potenzialmente capaci di generare un errore

In caso di divisione per zero viene generato un errore: in quel caso PERL riempie la variabile implicita \$@ con la stringa di caratteri che descrive il tipo di errore generato, in questo caso **Illegal division by zero**

Enter denominator: 7

Result: 2

Enter denominator: 0

Fatal!... Illegal division by zero at esempio_0_1.pl line 4.



PERL, come già detto, é perfettamente adatto anche per rappresentare e risolvere problemi di matematica, anzi offre molte interessanti proprietà: vediamo un esempio.

```
$x = [ [ 3, 2, 3 ], [ 5, 9, 8 ] ] ,
```

Un classico problema di moltiplicazione di matrici:

```
$y = [ [ 4, 7 ], [ 2, 3 ], [ 6, 1 ] ] ,
```

`$x` e `$y` contengono in realtà puntatori a delle strutture che contengono le matrici realizzate come vettori di vettori. Vedremo in seguito i dettagli e le implicazioni di questo concetto.

Se eseguiamo questo programma otteniamo:

```
$x * $y = [ [ 34 30 ] [ 86 70 ] ]
```

```
$z = mmult( $x, $y ) ;  
PrintMatrix( $z ) ;
```

L'implementazione del prodotto é codificata in questa funzione: contrariamente al FORTRAN, essa é in grado di determinare da sé il rango delle matrici e dimensionare opportunamente la matrice risultato (in questo caso `$z`)

La funzione `PrintMatrix` non fa altro che stampare il risultato `$z` in un formato opportuno, analogo a quello con cui sono definiti `$x` e `$y`



Un modo ancora più elegante e sintetico per fare questo tipo di calcolo, è quello di usare un modulo matematico, detto PDL, disponibile su WEB, che realizza il prodotto di matrici tramite la funzionalità di function-overloading (analoga quella del C++).

```
use PDL ;
```

```
$x = [
  [ 3, 2, 3 ],
  [ 5, 9, 8 ]
] ,
```

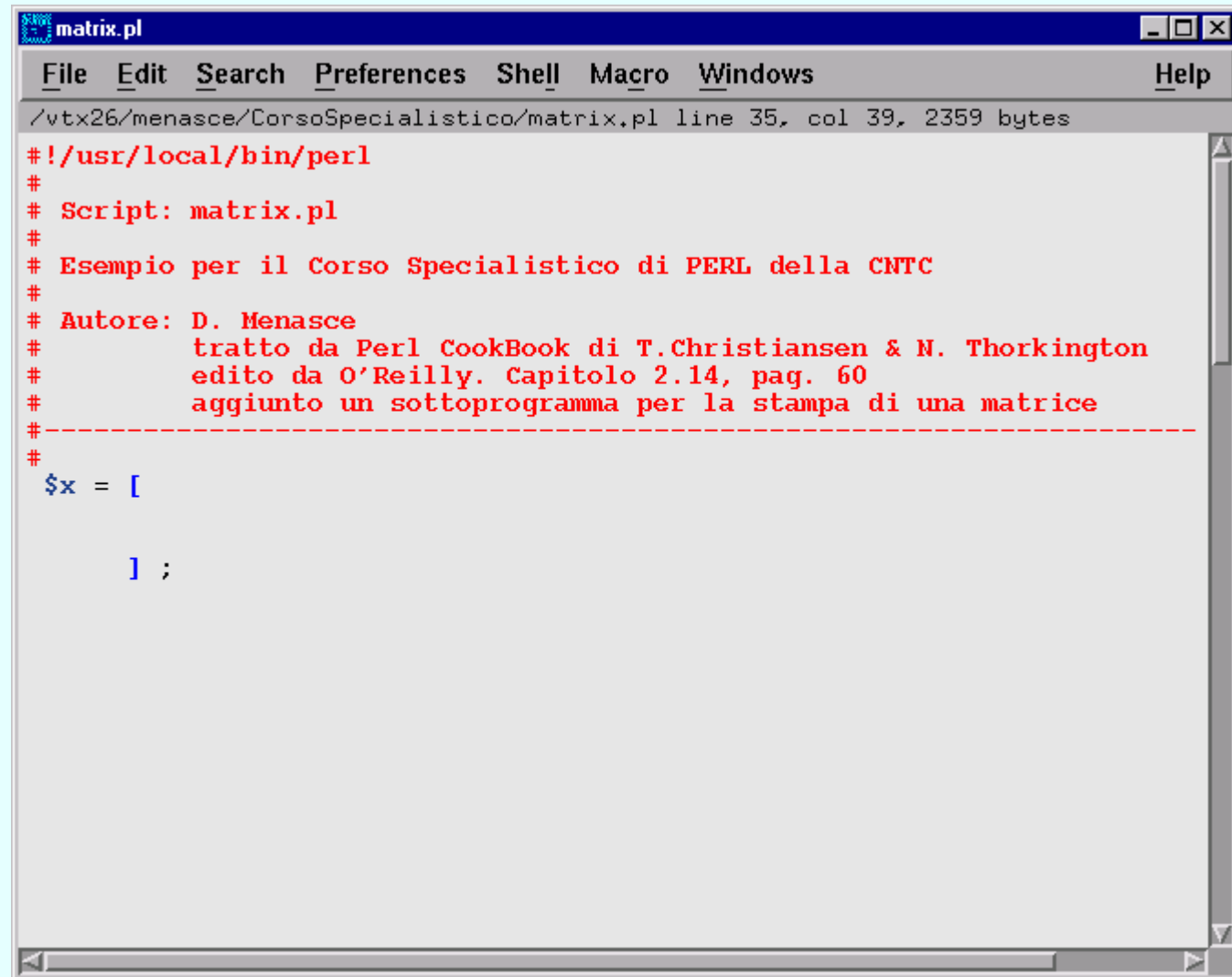
```
$y = [
  [ 4, 7 ],
  [ 2, 3 ],
  [ 6, 1 ]
] ,
```

```
$z = $x * $y ;
PrintMatrix( $z ) ;
```

L'inclusione del modulo PDL, fa sì che l'operatore `*` venga ridefinito in modo da agire sulle matrici `x` ed `y`, realizzando il prodotto righe per colonne.



Vediamo come si costruisce in PERL un programma di questo genere



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl line 35, col 39, 2359 bytes
#!/usr/local/bin/perl
#
# Script: matrix.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#        tratto da Perl Cookbook di T.Christiansen & N. Thorkington
#        edito da O'Reilly. Capitolo 2.14, pag. 60
#        aggiunto un sottoprogramma per la stampa di una matrice
#-----
#
$x = [
    ] ;
```



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl line 35, col 39, 2359 bytes
#!/usr/local/bin/perl
#
# Script: matrix.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#        tratto da Perl Cookbook di T.Christiansen & N. Thorkington
#        edito da O'Reilly. Capitolo 2.14, pag. 60
#        aggiunto un sottoprogramma per la stampa di una matrice
#-----
#
$x = [
    [ 3, 2, 3 ],
    [ 5, 9, 8 ]
];
```

Vogliamo scrivere le nostre matrici in modo naturale, come si fa tradizionalmente in algebra:



\$x e **\$y** contengono
le referenze a dei
vettori di vettori.

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl line 35, col 39, 2359 bytes
#!/usr/local/bin/perl
#
# Script: matrix.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#        tratto da Perl Cookbook di T.Christiansen & N. Thorkington
#        edito da O'Reilly. Capitolo 2.14, pag. 60
#        aggiunto un sottoprogramma per la stampa di una matrice
#-----
#
$x = [
    [ 3, 2, 3 ],
    [ 5, 9, 8 ]
];
$y = [
    [ 4, 7 ],
    [ 2, 3 ],
    [ 6, 1 ]
];
```



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl line 35, col 39, 2359 bytes
#!/usr/local/bin/perl
#
# Script: matrix.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#        tratto da Perl Cookbook di T.Christiansen & N. Thorkington
#        edito da O'Reilly. Capitolo 2.14, pag. 60
#        aggiunto un sottoprogramma per la stampa di una matrice
#-----
#
$x = [
    [ 3, 2, 3 ],
    [ 5, 9, 8 ]
];

$y = [
    [ 4, 7 ],
    [ 2, 3 ],
    [ 6, 1 ]
];

$z = mmult( $x, $y );
```



La funzione `mmult` dovrà operare, dal punto di vista concettuale, nel modo seguente:

The screenshot shows a Perl script editor window titled "matrix.pl" with a menu bar (File, Edit, Search, Preferences, Shell, Macro, Windows, Help) and a status bar showing the file path and size. The script content is as follows:

```
sub mmult {
```

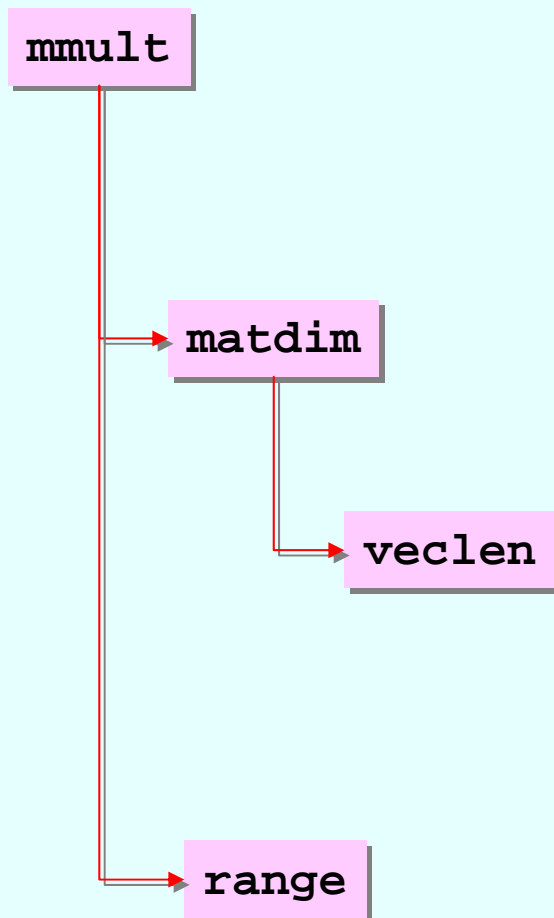
Three yellow callout boxes are overlaid on the script:

- Box 1: "Occorre preventivamente determinare il rango delle due matrici (ossia il numero di righe e di colonne)."
- Box 2: "Se le matrici non fossero di rango omologo, e quindi non fosse possibile eseguirne il prodotto, occorre generare un codice di errore e portare a termine il programma."
- Box 3: "Eeguire quindi il prodotto righe per colonne ciclando sugli indici delle matrici da 1 fino al valore del rango di ognuna di esse:"

To the right of the third callout box, the following mathematical formulas are displayed:

$$z_{i,j} = \sum_{k=1}^{n_k} x_{ik} \cdot y_{kj}$$
$$\forall i = 1, \dots, n_i,$$
$$\forall j = 1, \dots, n_j$$


Vediamo come organizzare l'implementazione di questo programma in uno schema concettuale a blocchi (ogni blocco rappresenta un opportuno sottoprogramma)



Date in ingresso le referenze a due matrici (definite come vettori di vettori), calcola una matrice risultante definita come il prodotto righe per colonne delle due matrici in ingresso

Data in ingresso la referenza ad una matrice (definita come vettore di vettori), ne determina il rango

Dato in ingresso la referenza ad un vettore ne determina il numero di elementi. Con due chiamate a questa funzione si calcolano rispettivamente il numero di righe e poi il numero di colonne di una matrice.

Dato in ingresso un numero n restituisce un pezzo di codice che rappresenta un range numerico nella forma $0 \dots n$. Questo serve per organizzare i cicli iterativi sul numero di elementi della matrice per eseguire il prodotto righe per colonne



Predisporremo infine una funzione per la stampa di una matrice, con cui rappresentare il risultato:

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl line 35, col 39, 2359 bytes
#!/usr/local/bin/perl
#
# Script: matrix.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#        tratto da Perl CookBook di T.Christiansen & N. Thorkington
#        edito da O'Reilly. Capitolo 2.14, pag. 60
#        aggiunto un sottoprogramma per la stampa di una matrice
#-----
#
$x = [
    [ 3, 2, 3 ],
    [ 5, 9, 8 ]
];

$y = [
    [ 4, 7 ],
    [ 2, 3 ],
    [ 6, 1 ]
];

$z = mmult( $x, $y );

&PrintMatrix( '$x', $x );
&PrintMatrix( '$y', $y );
&PrintMatrix( '$x * $y', $z );
```



Vediamo quindi come implementare concretamente la funzione `mmult`

```
$z = mmult( $x, $y );
```

Passo come argomenti a `mmult` le *referenze* alle rispettive strutture anonime, `$x` e `$y`

Determiniamo ora il *rango* delle due matrici in ingresso:

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

sub mmult {
  my ($m1,$m2) = @_;
}

}
```

Data una lista di argomenti forniti ad un sottoprogramma, questi vengono memorizzati nel vettore implicito di PERL `@_` che conterrà quindi in questo caso le due referenze `$x` e `$y` che vengono quindi trasferite rispettivamente in `$m1` e `$m2`



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes
sub mmult {
  my ($m1,$m2) = @_;
  my ($m1rows,$m1cols) = matdim($m1);
  my ($m2rows,$m2cols) = matdim($m2);
```

Data una matrice (più precisamente una referenza ad una matrice, `$m1`), determiniamone il **rango** mediante una chiamata al sottoprogramma `matdim`. Poiché le matrici in ingresso sono due (x ed y) dovremo predisporre due chiamate.



```
[ [3, 2, 3], [5, 9, 8] ]
```

`$m1`

```
matdim($m1);
```

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

    $x = [
        [3, 2, 3], vettore 1
        [5, 9, 8] vettore 2
    ]
                                $rows = 2

#=====

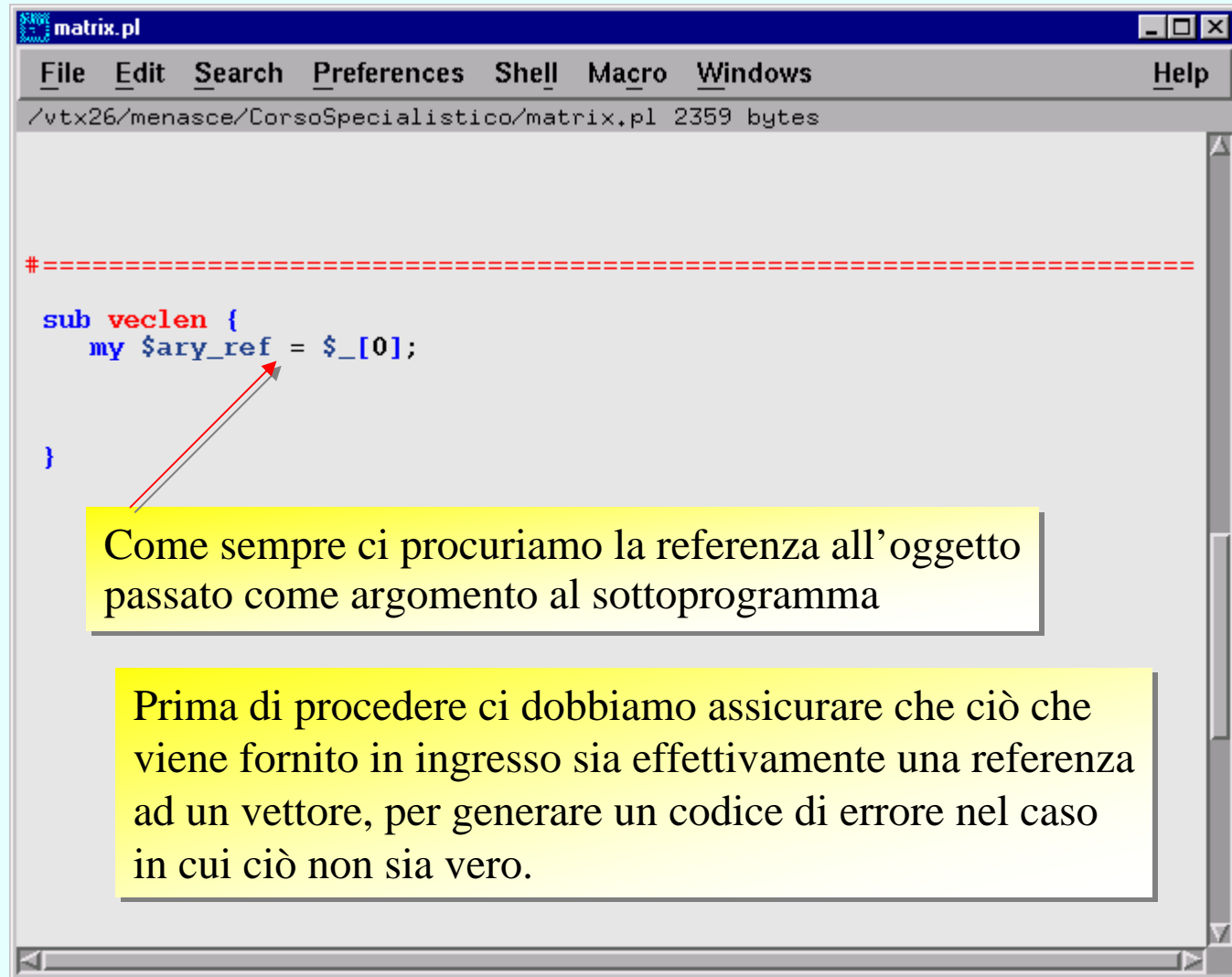
sub matdim {
    my $matrix = $_[0];
    my $rows = veclen($matrix);
}

#=====
```

Data una matrice (più precisamente una referenza ad una matrice, `$matrix`), determiniamone il numero di righe: ricordiamo che la matrice é rappresentata come vettore di vettori. Il numero di righe sarà quindi il numero di vettori contenuti in `$matrix`



Vediamo quindi come implementare concretamente la funzione **veclen**, la quale, data in ingresso una referenza ad una struttura composta da un vettore di vettori, ne determini il numero.



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

# =====

sub veclen {
  my $ary_ref = $_[0];
}


```

Come sempre ci procuriamo la referenza all'oggetto passato come argomento al sottoprogramma

Prima di procedere ci dobbiamo assicurare che ciò che viene fornito in ingresso sia effettivamente una referenza ad un vettore, per generare un codice di errore nel caso in cui ciò non sia vero.



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

# =====

sub veclen {
  my $ary_ref = $_[0];
  my $type = ref $ary_ref;
}

Data una reference, $ary_ref, l'operatore di PERL, ref, restituisce una stringa di caratteri che descrive di che tipo di variabile si tratta quella cui si riferisce il puntatore. Vediamo più in dettaglio come agisce questo importante operatore:
```



```
TypeOfVariable.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/TypeOfVariable.pl line 25, col 0, 431 bytes
#!/usr/local/bin/perl
#
# Script: TypeOfVariable.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
#
$a = "pippo";
@b = ("alfa", "beta") ;

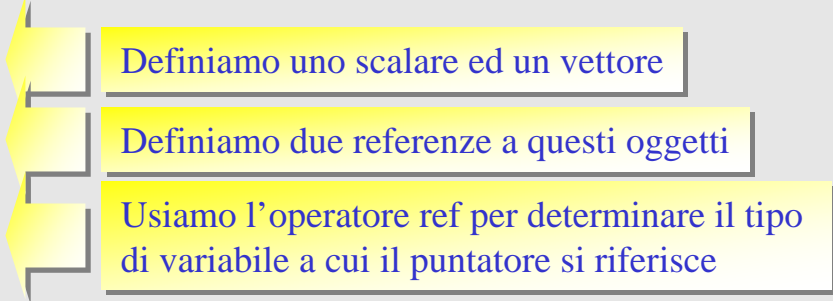
$ref_a = \$a ;
$ref_b = \@b ;

$type_a = ref $ref_a ;
$type_b = ref $ref_b ;

print <<EOT ;

La variable a e` di tipo $type_a
La variable b e` di tipo $type_b

EOT
```



ref é un operatore della famiglia degli *operatori introspettivi*: si tratta di oggetti di fondamentale utilità, in quanto permettono al codice di esaminare parti di se stesso e di agire in conseguenza.

```
TypeOfVariable
<CNTC> ./TypeOfVariable.pl

La variable a e` di tipo SCALAR
La variable b e` di tipo ARRAY

<CNTC> █
```



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

# =====

sub veclen {
  my $ary_ref = $_[0];
  my $type = ref $ary_ref;
  if ($type ne "ARRAY") { die "$type is bad array ref for $ary_ref" }
}


```

Predisponiamo una verifica che l'oggetto ricevuto in ingresso corrisponda effettivamente ad un **ARRAY** (questo ci permetterà in futuro di riutilizzare questo sottoprogramma come utile strumento in diverse circostanze)



`$ary_ref`

`$ary_ref` é la referenza al primo vettore dei due che costituiscono la matrice

`[3, 2, 3]`

`@$ary_ref`

`@$ary_ref` é il vettore stesso!

`scalar @array` restituisce la dimensione del vettore `@array`.

`scalar @$ary_ref` restituirà quindi la dimensione di **vettore 1** ossia **3**

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

# =====

sub veclen {
  my $ary_ref = $_[0];
  my $type = ref $ary_ref;
  if ($type ne "ARRAY") { die "$type is bad array ref for $ary_ref" }
  return scalar(@$ary_ref);
}

$x = [
  [3, 2, 3], vettore 1
  [5, 9, 8] vettore 2
]
```

A questo punto facciamo in modo che la funzione **veclen** restituisca in uscita la dimensione del vettore 1 calcolata nel modo appena indicato




```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

# =====

sub matdim {
  my $matrix = $_[0];
  my $rows = veclen($matrix);
  my $cols = veclen($matrix->[0]);
  return ($rows, $cols);
}

# =====
```

Per finire, il sottoprogramma **matdim** restituisce due scalari raggruppati in un vettore, **\$row** e **\$cols**, che rappresentano il rango di **\$matrix**



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes
sub mmult {
  my ($m1,$m2) = @_;
  my ($m1rows,$m1cols) = matdim($m1);
  my ($m2rows,$m2cols) = matdim($m2);

  unless ($m1cols == $m2rows) { # raise exception
    die "IndexError: matrices don't match: $m1cols != $m2rows";
  }
}
```

A questo punto **mmult** conosce il rango delle due matrici:

Verifichiamo che i ranghi delle due matrici siano tali per cui sia possibile il prodotto righe per colonne (ossia che un indice sia saturabile nella sommatoria)

$$z_{i,j} = \sum_{k=1}^{n_k} x_{ik} \cdot y_{kj}$$
$$\forall i = 1, \dots, n_i,$$
$$\forall j = 1, \dots, n_j$$



$$z_{i,j} = \sum_{k=1}^{n_k} x_{ik} \cdot y_{kj}$$

$$\forall i = 1, \dots, n_i,$$

$$\forall j = 1, \dots, n_j$$

```
for $i (1 .. 100) {  
}
```

equivale a $\sum_{i=1}^{100} i$

```
matrix.pl  
File Edit Search Preferences Shell Macro Windows Help  
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes  
  
sub mmult {  
  my ($m1,$m2) = @_  
  my ($m1rows,$m1cols) = matdim($m1);  
  my ($m2rows,$m2cols) = matdim($m2);  
  
  unless ($m1cols == $m2rows) { # raise exception  
    die "IndexError: matrices don't match: $m1cols != $m2rows";  
  }  
  
  for $i (range($m1rows)) {  
  
  }  
}
```

Vogliamo ora istituire un ciclo iterativo
che realizzi il costrutto $\forall i = 1, \dots, n_i$
Ci occorre un sottoprogramma che restituisca
un range (0 .. n_i-1)



`range ($m1rows)`

L'argomento di ingresso dell'operatore **range** é il numero di colonne della matrice

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes
#-----
sub range { 0 .. ($_[0] - 1) }
```

Il primo elemento della variabile implicita di **PERL**, `$_[0]`, corrisponde quindi al numero di colonne.

range restituisce quindi la stringa `0..2`

La funzione **range** restituisce un pezzo di codice **PERL** che può essere in seguito eseguito!!!



$$z_{i,j} = \sum_{k=1}^{n_k} x_{ik} \cdot y_{kj}$$

$$\forall i = 1, \dots, n_i,$$

$$\forall j = 1, \dots, n_j$$

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

sub mmult {
  my ($m1,$m2) = @_;
  my ($m1rows,$m1cols) = matdim($m1);
  my ($m2rows,$m2cols) = matdim($m2);

  unless ($m1cols == $m2rows) { # raise exception
    die "IndexError: matrices don't match: $m1cols != $m2rows";
  }

  for $i (range($m1rows)) {
    for $j (range($m2cols)) {
      for $k (range($m1cols)) {

      }
    }
  }
}
```

Il calcolo della sommatoria verrà realizzato all'interno di un triplo ciclo iterativo, uno per ogni indice mobile, **i**, **j** e **k**



$$z_{i,j} = \sum_{k=1}^{n_k} x_{ik} \cdot y_{kj}$$

$$\forall i = 1, \dots, n_i,$$

$$\forall j = 1, \dots, n_j$$

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

sub mmult {
  my ($m1,$m2) = @_;
  my ($m1rows,$m1cols) = matdim($m1);
  my ($m2rows,$m2cols) = matdim($m2);

  unless ($m1cols == $m2rows) { # raise exception
    die "IndexError: matrices don't match: $m1cols != $m2rows";
  }

  for $i (range($m1rows)) {
    for $j (range($m2cols)) {
      for $k (range($m1cols)) {
        $result->[$i][$j] += $m1->[$i][$k] * $m2->[$k][$j];
      }
    }
  }
}
```

L'elemento $z_{i,j}$ viene rappresentato da un vettore bi-dimensionale, `$result->[$i][$j]`



Dichiariamo locali le variabili usate all'interno di questo sottoprogramma

```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes

sub mmult {
  my ($m1,$m2) = @_;
  my ($m1rows,$m1cols) = matdim($m1);
  my ($m2rows,$m2cols) = matdim($m2);

  unless ($m1cols == $m2rows) { # raise exception
    die "IndexError: matrices don't match: $m1cols != $m2rows";
  }

  my $result = [];
  my ($i, $j, $k);

  for $i (range($m1rows)) {
    for $j (range($m2cols)) {
      for $k (range($m1cols)) {
        $result->[$i][$j] += $m1->[$i][$k] * $m2->[$k][$j];
      }
    }
  }
}
```




```
&PrintMatrix( '$x', $x ) ;  
&PrintMatrix( '$y', $y ) ;  
&PrintMatrix( '$x * $y', $z ) ;
```

```
matrix.pl  
File Edit Search Preferences Shell Macro Windows Help  
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2359 bytes  
  
sub PrintMatrix {  
  my $text = $_[0];  
  my $matrix = $_[1];  
  
}
```

Come al solito trasferiamo le variabili di ingresso al sottoprogramma in due variabili locali, rispettivamente **\$text** e **\$matrix**



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2310 bytes
sub PrintMatrix {
  my $text = $_[0];
  my $matrix = $_[1];
  my ($rows,$cols) = matdim($matrix);
}
}
```

Calcoliamo il numero di righe e di colonne della matrice in ingresso. Apparentemente questa é un'operazione ridondante (l'abbiamo già fatta in altri punti del programma): se però vogliamo che questo sottoprogramma sia autonomo (e quindi riutilizzabile in altri contesti) lo dobbiamo dotare di capacità introspettiva circa le variabili che gli vengono fornite in ingresso.



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2310 bytes

sub PrintMatrix {
  my $text = $_[0];
  my $matrix = $_[1];
  my ($rows,$cols) = matdim($matrix);

  foreach $row (range($rows)) {
    foreach $col (range($cols)) {
      print(" $matrix->[$row][$col]" );
    }
  }
}
```

Così come faremmo in FORTRAN, C od altri linguaggi, eseguiamo due cicli iterativi innestati per stampare i valori della matrice di ingresso.



Dotiamo poi il programma di quelle parti necessarie a formattare l'output prodotto in maniera esteticamente soddisfacente.

Definiamo una variabile contenente tanti spazi bianchi quanti sono i caratteri che compongono il testo in ingresso più 2

```
search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2310 bytes
sub PrintMatrix {
  my $text = $_[0];
  my $matrix = $_[1];
  my ($rows,$cols) = matdim($matrix);

  foreach $b (2 .. length( $text ) + 2) {$blank .= " " } ;

  foreach $row (range($rows)) {
    foreach $col (range($cols)) {
      print(" $matrix->[$row][$col]" ) ;
    }
  }
}

&PrintMatrix( '$x', $x ) ;
&PrintMatrix( '$y', $y ) ;
&PrintMatrix( '$x * $y', $z ) ;
```



```
matrix.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix.pl 2310 bytes

sub PrintMatrix {
  my $text = $_[0];
  my $matrix = $_[1];
  my ($rows,$cols) = matdim($matrix);

  my $row ;
  my $col ;
  my $blank = "" ;
  my $b ;

  foreach $b (2 .. length( $text ) + 2) {$blank .= " " } ;

  print("$text = [\n" ) ;

  foreach $row (range($rows)) {
    print("${blank} [" ) ;
    foreach $col (range($cols)) {
      print(" $matrix->[$row][$col]" ) ;
    }
    print(" ]\n" ) ;
  }
  print("${blank} ]\n\n" ) ;
}
}
```

Qualche print ...



E voilà...

```
matrix.pl
<CNTC> ./matrix.pl
$x = [
  [ 3 2 3 ]
  [ 5 9 8 ]
]

$y = [
  [ 4 7 ]
  [ 2 3 ]
  [ 6 1 ]
]

$x * $y = [
  [ 34 30 ]
  [ 86 70 ]
]

<CNTC> █
```

Il programma che abbiamo realizzato é in grado di moltiplicare matrici di qualsiasi rango senza che si debba modificare ogni volta il codice. Contrariamente al **FORTRAN** non dobbiamo prevedere una dimensione massima per le matrici in ingresso e per i vettori di servizio. Il programma, grazie alla capacità introspettive di **PERL**, si adatta alle nuove condizioni in modo autonomo.

In questo rudimentale esempio abbiamo utilizzato come elementi della matrice dei numeri interi. In realtà possiamo usare qualsiasi cosa, anche definita in modo implicito.

Supponiamo ad esempio di dover definire come elemento x_{12} della matrice, il numero di linee di programma del sorgente stesso del programma **matrix.pl**. Una prima domanda é: a che cavolo mi serve? Ma sulla possibile risposta sorvoliamo...

Una seconda domanda é: quante modifiche devo apportare? Risposta: due righe.



```
matrix2.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix2.pl 2440 bytes
#!/usr/local/bin/perl
#
# Script: matrix.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#        tratto da Perl CookBook di T.Christiansen & N. Thorkington
#        edito da O'Reilly. Capitolo 2.14, pag. 60
#        aggiunto un sottoprogramma per la stampa di una matrice
#-----
#
$x = [
  [ 3, 'cat matrix.pl | wc | awk '{print $1}'', 3 ],
  [ 5, 9, 8 ]
];

$Y = [
  [ 4, 7 ],
  [ 2, 3 ],
  [ 6, 1 ]
];

$z = mmult( $x, $Y );

&PrintMatrix( '$x', $x );
&PrintMatrix( '$Y', $Y );
&PrintMatrix( '$x * $Y', $z );
```



```
matrix2.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/matrix2.pl 2440 bytes

sub PrintMatrix {
my $text = $_[0];
my $matrix = $_[1];
my ($rows,$cols) = matdim($matrix);

my $row ;
my $col ;
my $blank = " " ;
my $b ;

foreach $b (2 .. length( $text ) + 2) {$blank .= " " } ;

print("$text = [\n" ) ;

foreach $row (range($rows)) {
print("${blank} [" ) ;
foreach $col (range($cols)) {
chomp( $matrix->[$row][$col] ) ;
print(" $matrix->[$row][$col]" ) ;
}
print(" ]\n" ) ;
}
print("${blank} ]\n\n" ) ;
}
}
```




```
menasce@almifome
<CNTC> ./matrix2.pl
$x = [
  [ 3 106 3 ]
  [ 5 9 8 ]
]

$y = [
  [ 4 7 ]
  [ 2 3 ]
  [ 6 1 ]
]

$x * $y = [
  [ 242 342 ]
  [ 86 70 ]
]

<CNTC> █
```

```
Shell Macro Windows Help
cico/matrix2.pl 2440 bytes

n($matrix);

$text ) + 2) {$blank .= " " } ;

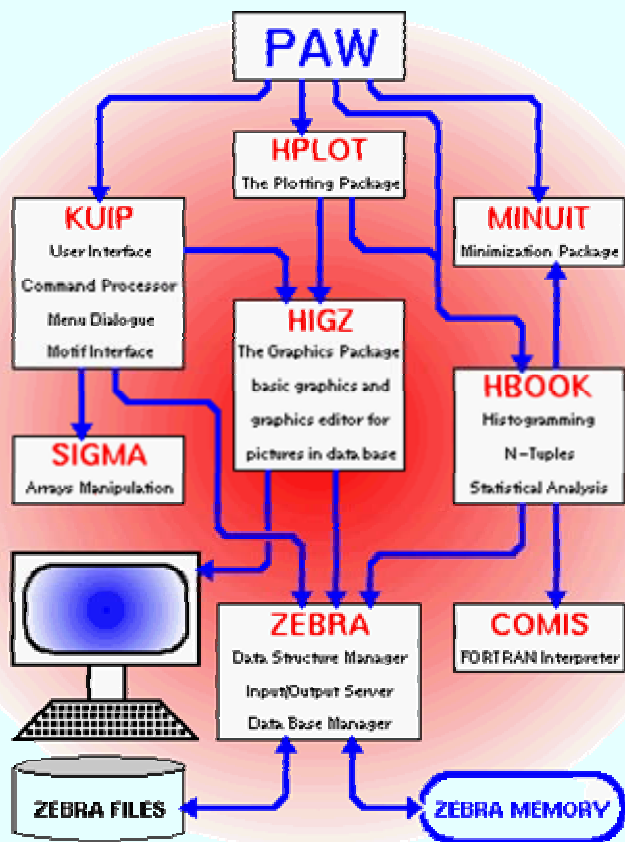
s)) {
s)) {
[$col] ) ;
[$col]" ) ;

print(" ]\n" ) ;
}
print("${blank} ]\n\n" ) ;
}
```



Passiamo ora ad esplorare le capacità di **PERL** in qualità di meta-linguaggio. Un ricercatore, sia quello impegnato in analisi dati che quello costretto ad un braccio di ferro con problemi di calibrazione di un rivelatore, ha frequenti necessità di interazione con **PAW** e la **CERNLIB**.

Chiunque abbia avuto un corpo a corpo con **PAW** si sarà reso conto di quanto sia utile e necessario ma nel contempo farraginoso nella sintassi ed oscuro nella implementazione (si tratta pur sempre di un coacervo di diversi linguaggi e delle corrispettive sintassi)



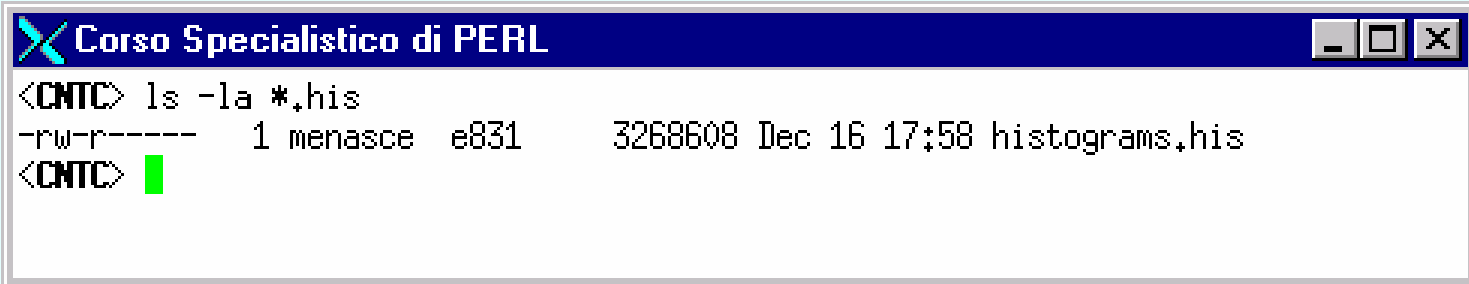
Vediamo come le proprietà di PERL ci forniscano la possibilità di semplificare grandemente l'utilizzo di PAW e addirittura di costruire oggetti sofisticati come un server di istogrammi interfacciato alla WEB.



Supponiamo di avere un file di istogrammi in formato **HBOOK**, `histograms.his`, e di voler produrre una lista degli istogrammi in esso contenuti. Poiché il formato **HBOOK** é basato su **ZEBRA**, il file suddetto é interpretabile unicamente da **PAW** o da un programma scritto dall'utente utilizzando i sottoprogrammi della libreria **CERNLIB** (non é possibile, in altre parole, dare un comando tipo `cat histograms.his` per leggere il contenuto del file).

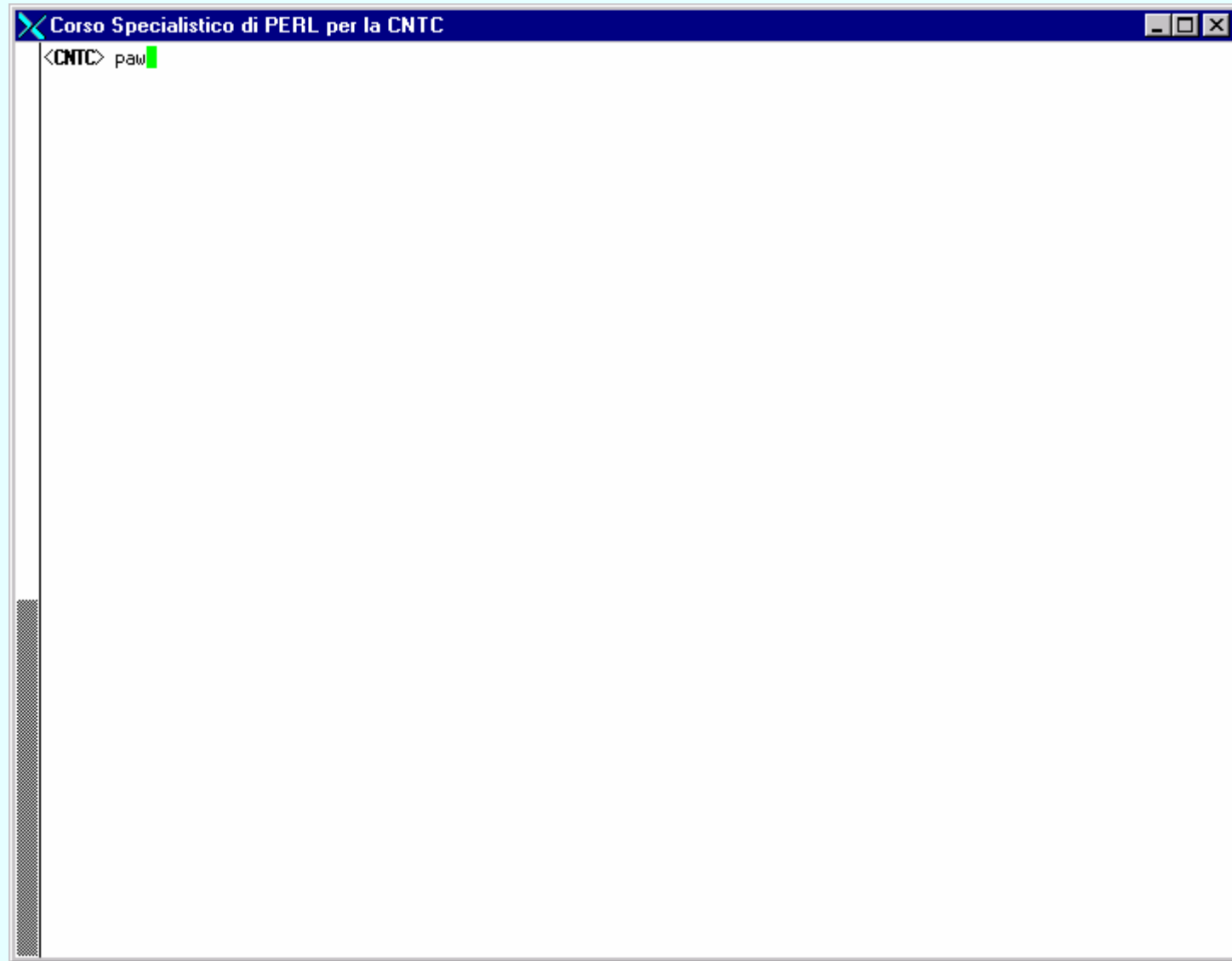
Questo é un problema tipico della programmazione. Certi dati sono accessibili unicamente da programmi specializzati nel manipolare i file che li contengono, ma tali programmi non possiedono certe funzionalità che ci occorrono. D'altra parte non possiamo modificare il sorgente del programma capace di leggere il file che ci interessa, sia perché la cosa é troppo complessa (questo é il caso della **CERNLIB**) o semplicemente perché non possediamo il codice sorgente.

Vediamo cosa fa tipicamente un utente che voglia leggere il contenuto di un file di istogrammi:



```
<CNTC> ls -la *.his
-rw-r-----  1 menasce  e831      3268608 Dec 16 17:58 histograms.his
<CNTC> █
```





```
<CNTC> paw
```



```
Curso Specialistico di PERL per la CNTC
<CNTC> paw
Calling 2000/IRIX-6-5 version of paw-X11
*****
*                                     *
*      W E L C O M E   to   P A W      *
*                                     *
*      Version 2.11/11      9 November 1999   *
*                                     *
*****
Workstation type (?=HELP) <CR>=1 : █
```



```
Corso Specialistico di PERL per la CNTC
<CNTC> paw
Calling 2000/IRIX-6-5 version of paw-X11
*****
*
*           W E L C O M E   to   P A W           *
*
*   Version 2.11/11       9 November 1999       *
*
*****
Workstation type (?=HELP) <CR>=1 ;
Version 1.26/04 of HIGZ started
*** No default PANLOGON file "/vtx26/menasce/.,pawlogon,kumac" found
PAW > █
```



```
Corso Specialistico di PERL per la CNTC
<CNTC> paw
Calling 2000/IRIX-6-5 version of paw-X11
*****
*
*           W E L C O M E   to   P A W           *
*
*      Version 2.11/11       9 November 1999     *
*
*****
Workstation type (?=HELP) <CR>=1 ;
Version 1.26/04 of HIGZ started
*** No default PANLOGON file "/vtx26/menasce/.,pawlogon,kumac" found
PAW > sh ls -la *.his
```



```
Curso Specialistico di PERL per la CNTC
<CNTC> paw
Calling 2000/IRIX-6-5 version of paw-X11
*****
*                               *
*      W E L C O M E   to   P A W      *
*                               *
*      Version 2.11/11      9 November 1999      *
*                               *
*****
Workstation type (?=HELP) <CR>=1 ;
Version 1.26/04 of HIGZ started
*** No default PANLOGON file "/vtx26/menasce/,pawlogon,kumac" found

PAW > sh ls -la *.his
-rw-r-----  1 menasce  e831      3268608 Dec 16 17:58 histograms.his
```




```
Corso Specialistico di PERL per la CNTC
<CNTC> paw
Calling 2000/IRIX-6-5 version of paw-X11
*****
*                               *
*      W E L C O M E   to   P A W      *
*                               *
*      Version 2.11/11      9 November 1999      *
*                               *
*****
Workstation type (?=HELP) <CR>=1 ;
Version 1.26/04 of HIGZ started
*** No default PANLOGON file "/vtx26/menasce/,pawlogon,kumac" found

PAW > sh ls -la *.his
-rw-r-----  1 menasce  e831      3268608 Dec 16 17:58 histograms.his
PAW > h/fil 1 histograms.his
```



```
Corso Specialistico di PERL per la CNTC
<CNTC> paw
Calling 2000/IRIX-6-5 version of paw-X11
*****
*
*           W E L C O M E   to   P A W           *
*
*       Version 2.11/11       9 November 1999   *
*
*****
Workstation type (?=HELP) <CR>=1 ;
Version 1.26/04 of HIGZ started
*** No default PANLOGON file "/vtx26/menasce/,pawlogon,kumac" found

PAW > sh ls -la *.his
-rw-r-----  1 menasce  e831    3268608 Dec 16 17:58 histograms.his
PAW > h/fil 1 histograms.his
PAW > h/lis
```



```
Corso Specialistico di PERL per la CNTC
<CNTC> paw
Calling 2000/IRIX-6-5 version of paw-X11
*****
*
*           W E L C O M E   to   P A W           *
*
*           Version 2.11/11       9 November 1999   *
*
*****
Workstation type (?=HELP) <CR>=1 ;
Version 1.26/04 of HIGZ started
*** No default PAWLOGON file "/vtx26/menasce/,pawlogon,kumac" found

PAW > sh ls -la *.his
-rw-r-----  1 menasce  e831      3268608 Dec 16 17:58 histograms.his
PAW > h/fil 1 histograms.his
PAW > h/lis

==> Directory :
  201 (2)  NR
 1201 (2)  Phases NR
  202 (2)  <[r](770)
 1202 (2)  Phases <[r](770)
  203 (2)  f?!(1275)
 1203 (2)  Phases f?!(1275)
  204 (2)  f?!(980)
 1204 (2)  Phases f?!(980)
  205 (2)  S?!(1475)
 1205 (2)  Phases S?!(1475)
  206 (2)  <[r](1450)
 1206 (2)  Phases <[r](1450)
  207 (2)  f?!(400)
 1207 (2)  Phases f?!(400)
  208 (2)  f?!(1300)
 1208 (2)  Phases f?!(1300)
  401 (2)  1
  402 (2)  2
 8000 (1)  Input file: master_fitter_generic_ds_ppp.his  Output file: ppp_effi_ds.his
```



Un altro approccio é quello di scrivere un programma in **FORTRAN** che faccia la stessa cosa:

Dichiarative di varia natura

Inizializzo HBOOK

Acquisisco il nome del file

Apro il file in lettura

Gestisco il codice di errore

Stampo l'elenco degli istogrammi

```
ShowHistograms.f
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.f line 41, col 0, 750 bytes

      Program ShowHistograms
      C
      C Programma: ShowHistogramsFull.f
      C
      C Esempio per il Corso Specialistico di PERL della CNTC
      C
      C Autore: D. Menasce
      C-----
      C
      C      Parameter (Memory = 1000000)
      C      Common /PAWC/ Hmemor( Memory )
      C
      C      Character*80      InputFileName
      C
      C      Data              Icycle / 0 /
      C-----
      C
      C      Call Hlimit( Memory )
      C
      C      Call Getarg( 1, InputFileName )
      C      Write( 6, * ) InputFileName
      C
      C      Call Hropen(      1,
      C      1              'Input',
      C      2              InputFileName,
      C      3              ' ',
      C      4              1024,
      C      5              Istatus )
      C
      C      If ( Istatus .Ne. 0 ) Then
      C      Write( 6, * ) 'Error opening file ', InputFileName
      C      Call Exit
      C      End If
      C
      C      Call Hrin( 0, Icycle, 0 )
      C      Call Hindex
      C
      C      Call Exit
      C      End
```



```
<CNTC> ./ShowHistograms histograms.his | more
histograms.his
```

1

```
.....
*
*   HBOOK   HBOOK   CERN           VERSION   4.21       HISTOGRAM AND PLOT INDEX           17/12/:0
*
*
*.....
*   NO              TITLE              ID  B/C  ENTRIES  DIM  NCHA   LOWER      UPPER      ADDRESS  LENGTH
*
*.....
```

NO	TITLE	ID	B/C	ENTRIES	DIM	NCHA	LOWER	UPPER	ADDRESS	LENGTH
1	NR	201	32	252873	2	X 180	0,000E+00	0,360E+01	996299	33167
						Y 180	0,000E+00	0,360E+01	963154	33136
2	Phases NR	1201	32	252873	2	X 180	0,000E+00	0,360E+01	963128	33169
						Y 180	0,000E+00	0,360E+01	929983	33136
3	<[r](770)	202	32	252873	2	X 180	0,000E+00	0,360E+01	929957	33169
						Y 180	0,000E+00	0,360E+01	896812	33136
4	Phases <[r](770)	1202	32	252873	2	X 180	0,000E+00	0,360E+01	896785	33170
						Y 180	0,000E+00	0,360E+01	863640	33136
5	f?!(1275)	203	32	252873	2	X 180	0,000E+00	0,360E+01	863614	33169
						Y 180	0,000E+00	0,360E+01	830469	33136
6	Phases f?!(1275)	1203	32	252873	2	X 180	0,000E+00	0,360E+01	830441	33171
						Y 180	0,000E+00	0,360E+01	797296	33136
7	f?!(980)	204	32	252873	2	X 180	0,000E+00	0,360E+01	797270	33169
						Y 180	0,000E+00	0,360E+01	764125	33136
8	Phases f?!(980)	1204	32	252873	2	X 180	0,000E+00	0,360E+01	764098	33170
						Y 180	0,000E+00	0,360E+01	730953	33136
9	S?!(1475)	205	32	252873	2	X 180	0,000E+00	0,360E+01	730927	33169
						Y 180	0,000E+00	0,360E+01	697782	33136
10	Phases S?!(1475)	1205	32	252873	2	X 180	0,000E+00	0,360E+01	697754	33171
						Y 180	0,000E+00	0,360E+01	664609	33136
11	<[r](1450)	206	32	252873	2	X 180	0,000E+00	0,360E+01	664583	33169
						Y 180	0,000E+00	0,360E+01	631438	33136
12	Phases <[r](1450)	1206	32	252873	2	X 180	0,000E+00	0,360E+01	631410	33171
						Y 180	0,000E+00	0,360E+01	598265	33136
13	f?!(400)	207	32	252873	2	X 180	0,000E+00	0,360E+01	598239	33169

Eseguo il programma **ShowHistograms** ed ottengo:



Se ciò che ci occorre é soltanto questo, o poco di più, entrambi questi approcci sono soddisfacenti (ed infatti sono utilizzati dalla quasi totale comunità dei ricercatori).

Supponiamo invece che sia nostro desiderio costruire un programma che faccia le seguenti cose:

- mostri la lista degli istogrammi contenuti in un file (se invocato con un argomento, consideri questo argomento come il nome del file, altrimenti proponga la lista di tutti i file presenti nella directory corrente affinché l'utente possa sceglierne uno da aprire).
- Presenti gli istogrammi trovati nel seguente ordine:
 - prima** gli istogrammi mono-dimensionali
 - poi** gli istogrammi bi-dimensionali (scatter plots)il tutto ordinandoli per **ID** crescente
- Dia la possibilità all'utente di selezionare uno degli istogrammi contenuti nel file affinché venga rappresentato graficamente sullo schermo, assieme ad una tabella numerica nella quale sia indicato, **bin per bin** il numero di **entries**, l'errore poissoniano ad esso associato ed il valore della **fit function** associata in quel punto (se l'istogramma dovesse contenere anche il risultato di un fit).

Si osserva facilmente che un programma di questo tipo é di difficoltosa scrittura, sia in **FORTRAN** che in **KUIP** (vedremo più avanti il perché di questa affermazione).



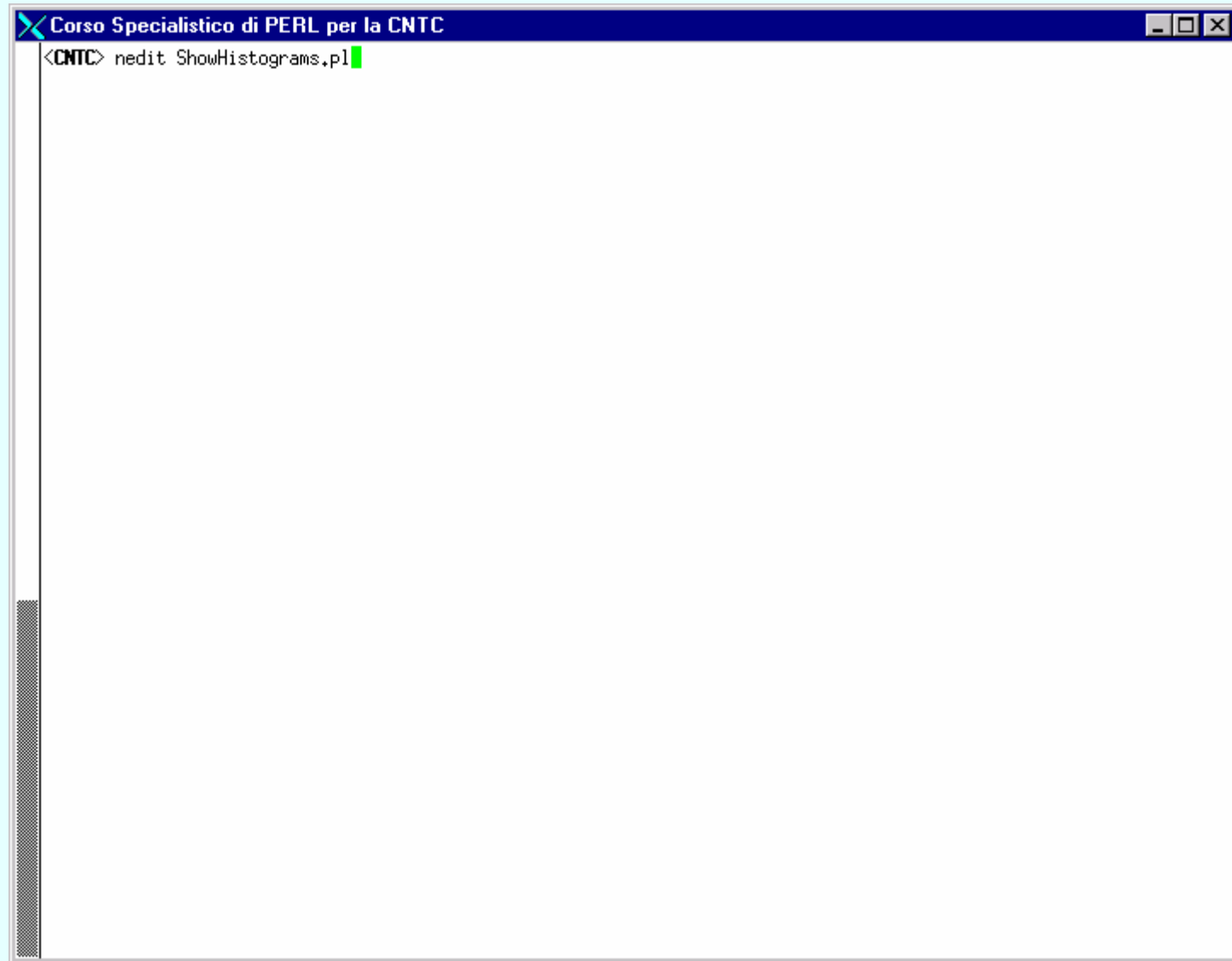
Vediamo come si può invece affrontare la risoluzione del problema utilizzando un altro approccio

Scriviamo un programma in **PERL** che invochi per noi **PAW**, e gli faccia eseguire tutti quei compiti di servizio per i quali **PAW** é ottimizzato, e si riservi invece la manipolazione di tutte quelle altre operazioni che avremmo difficoltà ad impostare sia in **FORTRAN** che in **KUIP**.

Utilizzeremo in questo modo **PERL** come **meta-linguaggio**, rendendolo capace di interagire con un altro programma (**PAW**), intercettandone l'output per ulteriori manipolazioni.

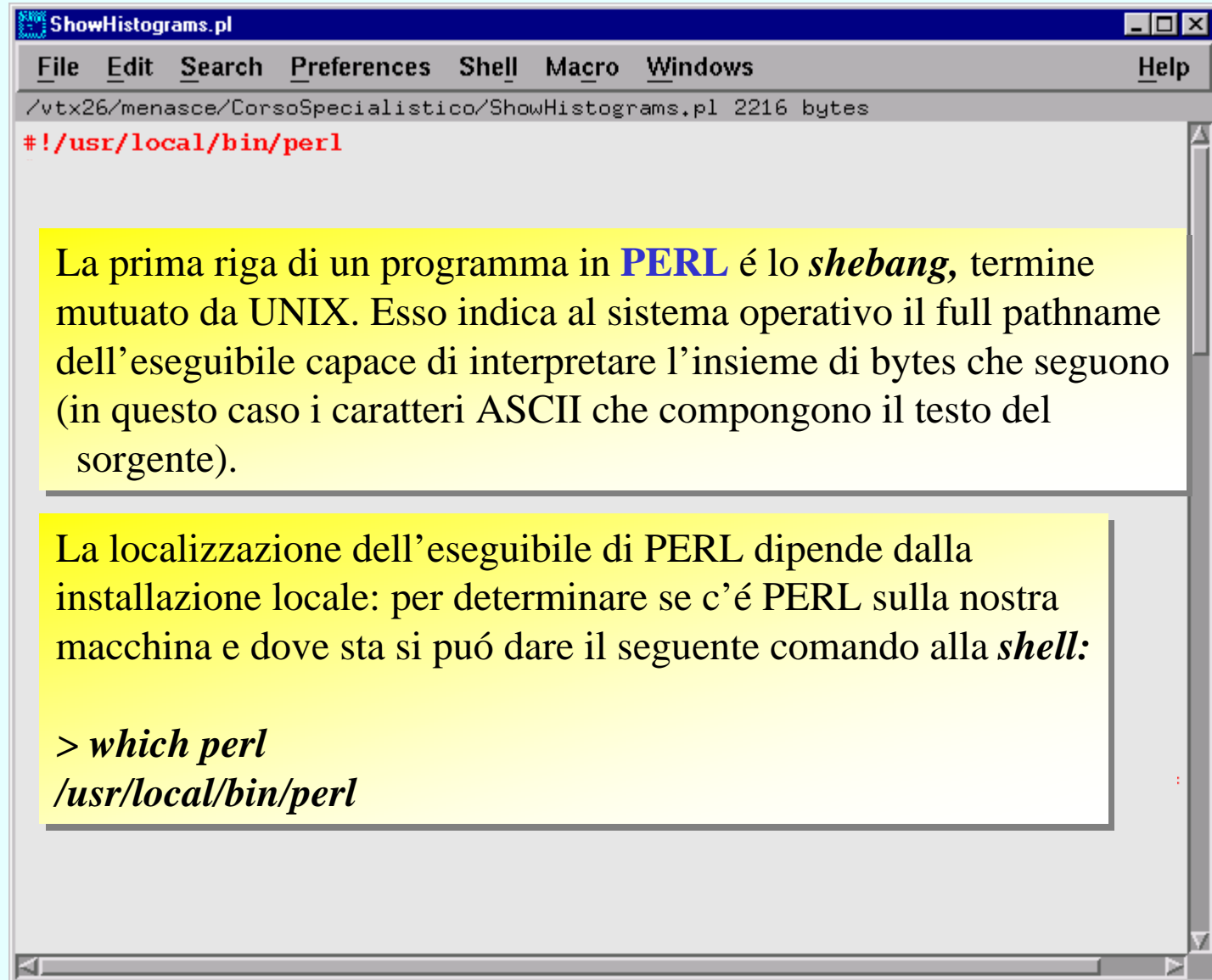
Così facendo avremo modo di renderci conto delle grandi capacità di **PERL** nel comportarsi come un collante fra procedure eterogenee e nel definire ed utilizzare potenti costrutti sintattici.





```
<CNTC> nedit ShowHistograms.pl
```





The image shows a screenshot of a text editor window titled "ShowHistograms.pl". The window has a menu bar with "File", "Edit", "Search", "Preferences", "Shell", "Macro", "Windows", and "Help". Below the menu bar, the file path is shown as "/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl" and the file size is "2216 bytes". The main content of the window is a single line of text: "#!/usr/local/bin/perl".

La prima riga di un programma in **PERL** é lo *shebang*, termine mutuato da UNIX. Esso indica al sistema operativo il full pathname dell'eseguibile capace di interpretare l'insieme di bytes che seguono (in questo caso i caratteri ASCII che compongono il testo del sorgente).

La localizzazione dell'eseguibile di PERL dipende dalla installazione locale: per determinare se c'è PERL sulla nostra macchina e dove sta si può dare il seguente comando alla *shell*:

```
> which perl  
/usr/local/bin/perl
```



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistograms.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

#-----
# Main body

$InputFileName = &GetHistogramFileName() ;
```

Lasciamo dello spazio per inizializzare delle variabili (quando occorrerà)

Variabile di ritorno:
il nome del file in questione.
(in questo caso uno scalare)

Nome di un sottoprogramma
(senza parametri in ingresso):
stabilisce il nome del file che
va aperto per elencarne il
contenuto sullo STANDARD
OUTPUT.



```
sub GetHistogramFileName {
```

Body del sottoprogramma

```
}
```



```
Corso Specialistico di PERL per la CNTC
<CNTC> ./ShowHistograms.pl histograms.his
```

ARGV[0]

Una regular-expression: un modo sintetico per rappresentare ed eventualmente selezionare un pattern di caratteri.

```
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes
sub GetHistogramFileName {
    if ( $ARGV[0] =~ m/\w+/ ) {
    } else {
    }
}
```

Se il primo elemento del vettore implicito di **PERL**, **ARGV**, contiene anche solo una lettera, significa che abbiamo invocato lo script con un argomento (il nome del file di istogrammi del quale vogliamo conoscere il contenuto).

Altrimenti prepariamoci a fornire all'utente l'elenco dei file, presenti nella directory corrente, il cui nome termina con suffisso **his** (nostra convenzione per indicare file di tipo **ZEBRA/HBOOK**).

```
Corso Specialistico di PERL per la CNTC
<CNTC> ./ShowHistograms.pl
```



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes
sub GetHistogramFileName {
    if ( $ARGV[0] =~ m/\w+/ ) {
        return $ARGV[0] ;
    } else {
        return $InputFileName ;
    }
}
}
```

Come parametro di ritorno, il corrente sottoprogramma restituirà il nome del file da leggere, preso *verbatim* dall'argomento di chiamata al programma

Se invece il nome l'avrà inserito l'utente (dopo aver visionato la lista che gli fornirà il pezzo di programma che ancora manca in questo punto) il sottoprogramma restituirà come valore di ritorno la variabile scalare che contiene tale nome (**\$InputFileName**)



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes

sub GetHistogramFileName {

    if ( $ARGV[0] =~ m/\w+/ ) {
        chomp( $ARGV[0] ) ;
        return $ARGV[0] ;
    } else {

        return $InputFileName ;
    }
}
}
```

Eliminiamo il carattere <CR> dal valore di input



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes

sub GetHistogramFileName {

    if ( $ARGV[0] =~ m/\w+/ ) {
        chomp( $ARGV[0] );
        return $ARGV[0] ;
    } else {
        $Argument = "*.his" ;

        foreach $file (glob $Argument) {
            print("$file\n" ) ;
        }

        chomp( $InputFileName = <STDIN> ) ;
        return $InputFileName ;
    }
}
}
```

L'operatore di **PERL**, **glob**, accetta come argomento lo stesso del comando **UNIX ls**: in questo caso ***.his**. **glob** fornisce in output una lista con tutti i nomi del file che soddisfano questo pattern proposto.

Una volta stampata la lista dei file di tipo istogramma esistenti nella directory corrente, il programma si aspetta come input dall'utente il nome del file desiderato



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistogramFileName {

    if ( $ARGV[0] =~ m/\w+/ ) {
        chomp( $ARGV[0] ) ;
        return $ARGV[0] ;
    } else {
        $Argument = "*.his" ;

        print("List of existing histogram files:\n\n" ) ;

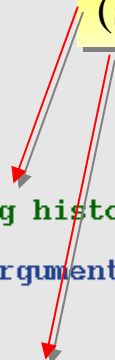
        foreach $file (glob $Argument) {
            print("$file\n" ) ;
        }

        print("\n\nPlease select a name: " ) ;

        chomp( $InputFileName = <STDIN> ) ;

        return $InputFileName ;
    }
}
}
```

Aggiungiamo delle frasi per migliorare (si fa per dire..) l'estetica dell'output




```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistogramFileName {

    system( "clear" ) ;
    if ( $ARGV[0] =~ m/\w+/ ) {
        chomp( $ARGV[0] ) ;
        return $ARGV[0] ;
    } else {

        $Argument = "*.his" ;

        print("List of existing histogram files:\n\n" ) ;

        foreach $file (glob $Argument) {
            print("$file\n" ) ;
        }

        print("\n\nPlease select a name: " ) ;

        chomp( $InputFileName = <STDIN> ) ;
        system( "clear" ) ;
        return $InputFileName ;

    }
}
}
```

L'operatore **system** accetta come argomento un qualsivoglia comando da delegare al soggiacente sistema operativo. In questo caso il comando **clear** di **UNIX** che cancella lo schermo (sempre questioni di estetica...)



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistogramFileName {

    my $Argument ;
    my $file ;

    system( "clear" ) ;

    if ( $ARGV[0] =~ m/\w+/ ) {

        chomp( $ARGV[0] ) ;
        return $ARGV[0] ;

    } else {

        $Argument = "*.his" ;

        print("List of existing histogram files:\n\n" ) ;

        foreach $file (glob $Argument) {
            print("$file\n" ) ;
        }

        print("\n\nPlease select a name: " ) ;

        chomp( $InputFileName = <STDIN> ) ;
        system( "clear" ) ;
        return $InputFileName ;

    }

}

}
```

Infine incapsuliamo le variabili `$Argument` e `$file` all'interno del corrente sottoprogramma con la dichiarativa `my` che ha l'effetto di rendere lo *scope* di una variabile locale al corpo del programma in cui é contenuta.





```
<CNTC> chmod +x ShowHistograms.pl
<CNTC> █
```



```
Corso Specialistico di PERL per la CNTC
<CNTC> chmod +x ShowHistograms.pl
<CNTC> ./ShowHistograms.pl
```



```
Corso Specialistico di PERL per la CNTC
List of existing histogram files:
histograms.his
Please select a name: █
```



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistograms.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

#-----
# Main body

$InputFileName = &GetHistogramFileName() ;
$EntitiesList = &GetHistograms("$InputFileName") ;
```

Scelto il nome del file da leggere, prepariamo un sottoprogramma che ci restituisca una opportuna struttura di dati con gli **ID** ed i titoli degli istogrammi trovati. Useremo questa struttura (descritta più oltre) per creare un report, opportunamente formattato, sul terminale. In questo caso vedremo che la variabile scalare `$EntitiesList` conterrà un **puntatore** (*pointer*) alla lista delle strutture create da `GetHistograms`



```
$EntitiesList = &GetHistograms("$InputFileName") ;
```

```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes
sub GetHistograms {
  my $InputFileName = shift ;
```

Gli argomenti passati ad un sottoprogramma vengono immagazzinati in una lista particolare, il *vettore implicito* di PERL, `@_`. L'operatore `shift` di PERL, estrae dalla lista `@_` un elemento alla volta (essendo una variabile implicita non occorre specificarne il nome).



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes
sub GetHistograms {
    my $InputFileName = shift ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;
```

Apriamo un file in scrittura: esso conterrà la macro **KUIP** che ci occorre.

Vogliamo ora che il nostro programma scriva una procedura **KUIP**, che poi eseguirà, di questo tipo:

```
Macro TempFile

H/File 1 nome-di-un-file-istogrammi
H/Lis
```




```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistograms.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

$MakeListKumac = "MakeList.kumac" ;
#-----
# Main body

$InputFileName = &GetHistogramFileName() ;
$EntitiesList = &GetHistograms("$InputFileName") ;
```

```
Help
eListKumac $!" ;
```

Definiamo una variabile.
Essa contiene il nome del file con
la macro **KUIP** che il nostro script
dovrà eseguire per avere la lista
degli istogrammi desiderata.



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistograms {
    my $InputFileName = shift ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;
    print OUT <<EOT ;
    Macro TempFile
        H/File 1 $InputFileName
        H/Lis
    EOT
    close( OUT ) ;
}
```

Scriviamo il testo della macro nel file che abbiamo precedentemente aperto. Notiamo come il nome del file sia una variabile **PERL** che, per quanto riguarda **KUIP** sarà una stringa di caratteri costante.



Vogliamo ora invocare **paw** (in modalità **batch**) per eseguire il file con la macro **KUIP** appena definita. Utilizziamo il concetto di **pipe**:

```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistograms {

    my $InputFileName = shift ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

    print OUT <<EOT ;
Macro TempFile

        H/File 1 $InputFileName
        H/Lis

EOT

    close( OUT ) ;

    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {

        }

}
```



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl line 93, col 0, 2048 bytes
sub GetHistograms {
    my $InputFileName = shift ;

    H/File 1 $InputFileName
    H/Lis

    EOT

    close( OUT ) ;
    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {
        print ;
    }
}
```

Man mano che il comando di **pipe** viene eseguito, lo **STANDARD OUTPUT** viene filtrato dal canale di comunicazione aperto (nel nostro caso **<CMD>**). Ogni singola riga viene quindi mandata al terminale mediante un semplice comando di **print**.





```
Corso Specialistico di PERL per la CNTC
<CNTC> ./ShowHistograms.pl histograms.his
```



```
Corso Specialistico di PERL per la CNTC
<CNTC> ./ShowHistograms.pl histograms.his

Calling 2000/IRIX-6-5 version of paw-X11

Version 1.26/04 of HIGZ started
*** No default PANLOGON file "/vtx26/menasce/.pawlogon.kumac" found

===> Directory :
 201 (2) NR
1201 (2) Phases NR
 202 (2) <[r](770)
1202 (2) Phases <[r](770)
 203 (2) f?!(1275)
1203 (2) Phases f?!(1275)
 204 (2) f?!(980)
1204 (2) Phases f?!(980)
 205 (2) S?!(1475)
1205 (2) Phases S?!(1475)
 206 (2) <[r](1450)
1206 (2) Phases <[r](1450)
 207 (2) f?!(400)
1207 (2) Phases f?!(400)
 208 (2) f?!(1300)
1208 (2) Phases f?!(1300)
 401 (2) 1
 402 (2) 2
8000 (1) Input file: master_fitter_genric_ds_ppp.his Output file: ppp_effi_ds.his
1313 (1) Dalitz Dati
1314 (1) Dalitz Dati
1510 (2) Dalitz Dati
1514 (2) Dalitz Effi Check
2001 (1) <[PPP] mass (Montecarlo)
1503 (2) ds side bands
1501 (2) ds model
1000 (1) Full parameters space description
3201 (1) NR
3202 (1) <[r](770)
3203 (1) f?!(1275)
3204 (1) f?!(980)
```

Istogrammi bi-dimensionali
(scatter plots)

Vogliamo ora stampare **prima** la lista
di tutti gli istogrammi bi-dimensionali
e **dopo** quella dei mono-dimensionali



```

ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistograms {

    my $InputFileName = shift ;

    my $ID      ;
    my $Type    ;
    my $Title   ;
    my $Name    ;
    my @Types   ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

    print OUT <<EOT ;
Macro TempFile

        H/File 1 $InputFileName
        H/Lis

EOT

    close( OUT ) ;

    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {
        if ( m/\s+(\d+)\s+\((\d+)\)\s+(.*)/ ) {
            1201 (2) Phases NR
        }
    }
}

```

Definiamo allo scopo un criterio di selezione basato su una **regexp** di una certa complessità, capace di discriminare, nel flusso di output, il seguente pattern di caratteri:



```

ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistograms {

    my $InputFileName = shift ;

    my $ID      ;
    my $Type   ;
    my $Title  ;
    my $Name   ;
    my @Types  ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

    print OUT <<EOT ;
Macro TempFile
        H/File 1 $InputFileName
        H/Lis
EOT
        1201 (2) Phases NR
    close( OUT ) ;

    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {
        if ( m/\s+(\d+)\s+\([\d+]\)\s+(\. *)/ ) {
            $ID      = $1 ;
            $Type   = $2 ;
            $Title  = $3 ;
        }
    }
}

```



Vogliamo ora creare una struttura di dati che associ allo **ID** di un istogramma il titolo dell'istogramma stesso, mantenendo però distinti gli istogrammi **mono**-dimensionali da quelli **bi**-dimensionali.

La struttura adatta allo scopo si chiama **HASH** ed assomiglia ad un convenzionale **ARRAY** (ad esempio in **FORTRAN** od in **C**). La differenza fondamentale é che l'indice di una **HASH** può essere una qualsiasi entità e non solo un numero intero come negli **ARRAY**.

Vogliamo quindi costruire due **HASH**, chiamate rispettivamente **Mono** e **Bi** che abbiano la seguente struttura:

```
$Bi{201}      = "NR" ;  
$Bi{1201}    = "Phases NR" ;  
...  
$Mono{8000}  = "Input file: master_fitter_genric_ds_ppp.his ..." ;  
$Mono{1313}  = "Dalitz dati" ;
```

```
==> Directory :  
   201 (2)  NR  
  1201 (2)  Phases NR  
   202 (2)  <[r](770)  
   ...  
  8000 (1)  Input file: master_fitter_genric_ds_ppp.his  Output file: ppp_effi_ds.his  
  1313 (1)  Dalitz Dati  
   ...
```



Vogliamo però costruire il programma con un sufficiente grado di generalità affinché nel futuro sia estendibile a riconoscere anche gli **ID** delle **n-tuple**, senza dovere infarcire il programma di strutture condizionali. Estendere un programma mediante l'aggiunta di nuove strutture condizionali é certamente sempre possibile ma non é la strategia migliore. Vediamo perché:

Un approccio al problema potrebbe essere il seguente:

```
...
if ( $Type == 1 ) {
    $Mono{$ID} = $Title ;
} else {
    $Bi{$ID}   = $Title ;
}
...
```

Se volessimo considerare anche il caso di **ID** di **n-tuple** dovremmo aggiungere un altro **elsif**. Questo per ogni nuovo tipo di categoria noi volessimo aggiungere in seguito.



In **PERL** esiste invece la possibilità di rendere variabile anche il nome di una variabile! Il blocco di istruzioni qui a destra diverrebbe, usando questo costrutto, una sola riga di codice:

```
...
$$ {Name} { $ID } = $Title ;
...
```

Dove **\$Name** vale **Bi** o **Mono** a seconda di **\$Type**.

```
...
if (      $Type == 1 ) {
    $Mono{$ID} = $Title ;
} elsif ( $Type == 2 ) {
    $Bi{$ID}   = $Title ;
} elsif ( $Type == 3 ) {
    $Ntuple{$ID} = $Title ;
}
...
```



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2216 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistograms.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations
$Type{1} = "mono" ;
$Type{2} = "bi" ;
$MakeListKumac = "MakeList.kumac" ;
#-----
# Main body

$InputFileName = &GetHistogramFileName() ;
$EntitiesList = &GetHistograms("$InputFileName") ;
```

Definiamo una opportuna HASH.

`$Type{1} = "mono" ;`
`$Type{2} = "bi" ;`



Osserviamo come un costrutto del tipo `$Type{$Type}` sia assolutamente legittimo in PERL.

Il primo `$Type` é il nome di una hash mentre il secondo é il nome di uno scalare. **PERL** sa riconoscere il tipo di variabile dal contesto (si dice che i *namespace* per i differenti tipi di variabili sono tenuti separati).

```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistograms {

    my $InputFileName = shift ;

    my $ID      ;
    my $Type    ;
    my $Title   ;
    my $Name    ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

    print OUT <<EOT ;
Macro TempFile

        H/File 1 $InputFileName
        H/Lis

EOT

    close( OUT ) ;

    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {
        if ( m/\s+(\d+)\s+\((\d+)\)\s+(.*)/ ) {
            $ID      = $1 ;
            $Type    = $2 ;
            $Title   = $3 ;
            $Name    = $Type{$Type} ;
        }
    }
}
```

`$Type=1` ⇒ `$Name=$Type{1}="Mono"` ;
`$Type=2` ⇒ `$Name=$Type{2}="Bi"` ;



```

ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistograms {

    my $InputFileName = shift ;

    my $ID      ;
    my $Type   ;
    my $Title  ;
    my $Name   ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

    print OUT <<EOT ;
Macro TempFile
        1201 (2) Phases NR
H/File 1 $InputFileName
H/Lis
EOT
    close( OUT ) ;

    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {
        if ( m/\s+(\d+)\s+\((\d+)\)\s+(.*)/ ) {
            $ID      = $1 ;
            $Type    = $2 ;
            $Title   = $3 ;
            $Name    = $Type{$Type} ;
            $$Name{$ID} = $Title ;
        }
    }
}

```



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2215 bytes

sub GetHistograms {

    my $InputFileName = shift ;

    my $ID      ;
    my $Type    ;
    my $Title   ;
    my $Name    ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

    print OUT <<EOT ;
Macro TempFile

        H/File 1 $InputFileName
        H/Lis

EOT

    close( OUT ) ;

    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {
        if ( m/\s+(\d+)\s+\((\d+)\)\s+(.*)/ ) {
            $ID      = $1 ;
            $Type    = $2 ;
            $Title   = $3 ;
            $Name    = $Type{$Type} ;
            $$Name{$ID} = $Title ;
            $TypeNames{$Name}++ ;
        }
    }
}
```

Contiamo il numero di differenti tipi di istogrammi trovati: allo scopo incrementiamo il contenuto di una opportuna hash mediante l'operatore di autoincremento ++



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2213 bytes

my $Title ;
my $Name ;

open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

print OUT <<EOT ;
Macro TempFile

    H/File 1 $InputFileName
    H/Lis

EOT

close( OUT ) ;

open( CMD, "paw -b $MakeListKumac | " ) ;
while( <CMD> ) {
    if ( m/\s+(\d+)\s+\((\d+)\)\s+(.*)/ ) {
        $ID      = $1 ;
        $Type    = $2 ;
        $Title   = $3 ;
        $Name    = $Type{$Type} ;
        $$ {Name} {ID} = $Title ;
        $TypeNames{$Name}++ ;
    }
}
close( CMD ) ;

return \%TypeNames ;

}
```

Infine, questa funzione ritorna al programma chiamante la referenza alla hash che contiene il numero di tipi differenti di strutture trovate nel file. Notiamo come le hash che contengono l'associazione ID con titolo (per ogni tipo differente) siano di tipo globale (non dichiarate **my**)



Lo invociamo passandogli la referenza alla struttura di dati prodotta da GetHistograms, la hash le cui chiavi sono i tipi di istogrammi trovati.

```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl 2213 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistograms.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations
$Type{1} = "mono" ;
$Type{2} = "bi" ;
$MakeListKumac = "MakeList.kumac" ;
#-----
# Main body
$InputFileName = &GetHistogramFileName() ;
$EntitiesList = &GetHistograms("$InputFileName") ;
print("List of histograms found in $InputFileName\n\n") ;
&ShowDirectory($EntitiesList) ;
# End of main body.
#-----
```

A questo punto chiamiamo un sottoprogramma il cui scopo sia quello di formattare una pagina con il contenuto trovato nel file richiesto.




```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl line 137, col 0, 2247 bytes
sub ShowDirectory {
    my $EntitiesList = shift ;

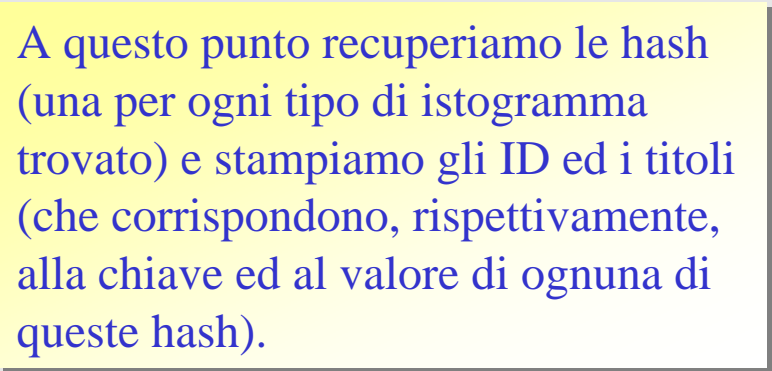
    foreach $HisType (keys %$EntitiesList) {
        print("List of ${HisType}-dimensional histograms\n\n" ) ;
    }
}
}
```

Cicliamo sugli elementi della hash in ingresso, e stampiamo le chiavi trovate (i tipi di istogrammi presenti nel file).



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl line 137, col 0, 2247 bytes
sub ShowDirectory {
    my $EntitiesList = shift ;

    foreach $HisType (keys %$EntitiesList) {
        print("List of ${HisType}-dimensional histograms\n\n" ) ;
        foreach $ID (keys %$HisType ) {
            printf("%5d %s\n", $ID, $$HisType{$ID} ) ;
        }
    }
}
```



A questo punto recuperiamo le hash (una per ogni tipo di istogramma trovato) e stampiamo gli ID ed i titoli (che corrispondono, rispettivamente, alla chiave ed al valore di ognuna di queste hash).



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl line 137, col 0, 2247 bytes
sub ShowDirectory {
    my $EntitiesList = shift ;

    { my $HisType ;
      my $ID ;

      foreach $HisType (keys %$EntitiesList) {
          print("List of ${HisType}-dimensional histograms\n\n" ) ;
          foreach $ID (keys %$HisType ) {
              printf("%5d %s\n", $ID, $$HisType{$ID} ) ;
          }
      }
    }
}
```

Rendiamo locali le variabili di uso privato a questa funzione.



```

X Corso Specialistico di PERL per la CNTC
List of histograms found in histograms,his

List of mono-dimensional histograms

1000 Full parameters space description
2001 <[PPP] mass (Montecarlo)
3201 NR
3202 <[r](770)
8000 Input file: master_fitter_genric_ds_ppp,his  Output file: ppp_effi_ds,his
1313 Dalitz Dati
3203 f??!(1275)
1314 Dalitz Dati
3204 f?0!(980)
3205 S?0!(1475)
1505 Initial MiniMC Parameters
3206 <[r](1450)
1506 Fit      MiniMC Parameters
3207 f?0!(400)
1507 Fit Informations
3208 f?0!(1300)

List of bi-dimensional histograms

1201 Phases NR
1202 Phases <[r](770)
1203 Phases f??!(1275)
1501 ds model
1510 Dalitz Dati
1204 Phases f?0!(980)
1205 Phases S?0!(1475)
1503 ds side bands
1206 Phases <[r](1450)
1504 Fit Dalitz function (S+B)
1207 Phases f?0!(400)
1514 Dalitz Effi  Check
1208 Phases f?0!(1300)
1515 Dalitz Adaptive Binning
1516 Pure background function from full fit
1517 Fit Dalitz function (Non physical)
1518 Dalitz data (Non physical)
1519 Fit Dalitz function (S)
201 NR
202 <[r](770)
401 1
203 f??!(1275)
402 2
204 f?0!(980)
205 S?0!(1475)
206 <[r](1450)
207 f?0!(400)
208 f?0!(1300)
<CNTC> █

```

Prima i mono dimensionali

Poi i bi dimensionali

E se volessimo questo elenco in ordine crescente di ID di istogramma?

Basta una banalissima modifica ad una sola riga del codice:



```
ShowHistograms.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistograms.pl line 137, col 0, 2264 bytes
sub ShowDirectory {
    my $EntitiesList = shift ;

    my $HisType ;
    my $ID ;

    foreach $HisType (keys %$EntitiesList) {
        print("\nList of ${HisType}-dimensional histograms\n\n" ) ;
        foreach $ID (sort {$a <=> $b} keys %$HisType ) {
            printf("%5d %s\n", $ID, $$HisType{$ID} ) ;
        }
    }
}

}

sort {$a <=> $b}
```



```
Corso Specialistico di PERL per la CNTC
List of histograms found in histograms.his

List of mono-dimensional histograms

1000 Full parameters space description
1313 Dalitz Dati
1314 Dalitz Dati
1505 Initial MiniMC Parameters
1506 Fit MiniMC Parameters
1507 Fit Informations
2001 <[PPP] mass (Montecarlo)
3201 NR
3202 <[r](770)
3203 f?2!(1275)
3204 f?0!(980)
3205 S?0!(1475)
3206 <[r](1450)
3207 f?0!(400)
3208 f?0!(1300)
8000 Input file: master_fitter_genric_ds_ppp.his Output file: ppp_effi_ds.his

List of bi-dimensional histograms

201 NR
202 <[r](770)
203 f?2!(1275)
204 f?0!(980)
205 S?0!(1475)
206 <[r](1450)
207 f?0!(400)
208 f?0!(1300)
401 1
402 2
1201 Phases NR
1202 Phases <[r](770)
1203 Phases f?2!(1275)
1204 Phases f?0!(980)
1205 Phases S?0!(1475)
1206 Phases <[r](1450)
1207 Phases f?0!(400)
1208 Phases f?0!(1300)
1501 ds model
1503 ds side bands
1504 Fit Dalitz function (S+B)
1510 Dalitz Dati
1514 Dalitz Effi Check
1515 Dalitz Adaptive Binning
1516 Pure background function from full fit
1517 Fit Dalitz function (Non physical)
1518 Dalitz data (Non physical)
1519 Fit Dalitz function (S)
<CNTC> █
```

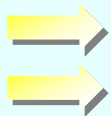


Abbiamo ora a disposizione un rudimentale *browser* di file di istogrammi.

Le funzioni sono però limitate alla sola capacità di ispezione e riordinamento degli ID e corrispondenti titoli. Vogliamo certamente qualcosina di più, come ad esempio, la possibilità, dato l'elenco, di selezionare un istogramma in particolare, e di visualizzarlo sullo schermo, magari con il dettaglio del contenuto stampato in maniera elegante (si fa per dire...).



Come primo passo, doteremo la funzione ShowDirectory della possibilità di fornire, a richiesta dell'utente, un ID di istogramma.



Creeremo poi una funzione per la stampa dell'istogr. selezionato sullo schermo

```
ShowHistogramsFull.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsFull.pl 4239 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistogramsFull.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations
$Type{1} = "mono" ;
$Type{2} = "bi" ;
$MakeListKumac = "MakeList.kumac" ;
#-----
# Main body
$InputFileName = &GetHistogramFileName() ;
$EntitiesList = &GetHistograms("$InputFileName") ;
print("List of histograms found in $InputFileName\n\n") ;
$IDToDisplay = &ShowDirectory($EntitiesList) ;
&PrintHistogram("$InputFileName", "$IDToDisplay") ;
# End of main body.
#=====
```




```
ShowHistogramsFull.pl
File Edit Search Preferences Shell Macro Windows Help
'vtx26/menasce/CorsoSpecialistico/ShowHistogramsFull.pl line 148, col 18, 4239 bytes
sub ShowDirectory {
    my $EntitiesList = shift ;

    foreach $HisType (keys %$EntitiesList) {
        print("\nList of ${HisType}-dimensional histograms\n\n" ) ;
        foreach $ID (sort {$a <=> $b} keys %$HisType ) {
            printf("%5d %s\n", $ID, $$HisType{$ID} ) ;
        }
    }

    print("\n\nSelect an ID: " ) ;
    chomp( $ID = <STDIN> ) ;
    $ID =~ s/\s+//g ; # Remove unwanted space-bar characters
    return "$ID" ;
}

#=====
```

A questo punto del corso non dovrebbe più essere necessario commentare ulteriormente questo pezzo di codice per comprenderne le funzionalità....



```
sub PrintHistogram {  
    my $InputFileName = shift ;  
    my $ID             = shift ;  
}
```

Acquisiamo in variabili locali *nome* del file ed *Id* dell'istogramma da stampare sul video.



```
sub PrintHistogram {  
    my $InputFileName = shift ;  
    my $ID             = shift ;  
  
    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;  
}
```



```
ShowHistogramsFull.pl
File Edit Search Preferences Shell Macro Windows Help
'vtx26/menasce/CorsoSpecialistico/ShowHistogramsFull.pl 4239 bytes

sub PrintHistogram {
    my $InputFileName = shift ;
    my $ID             = shift ;

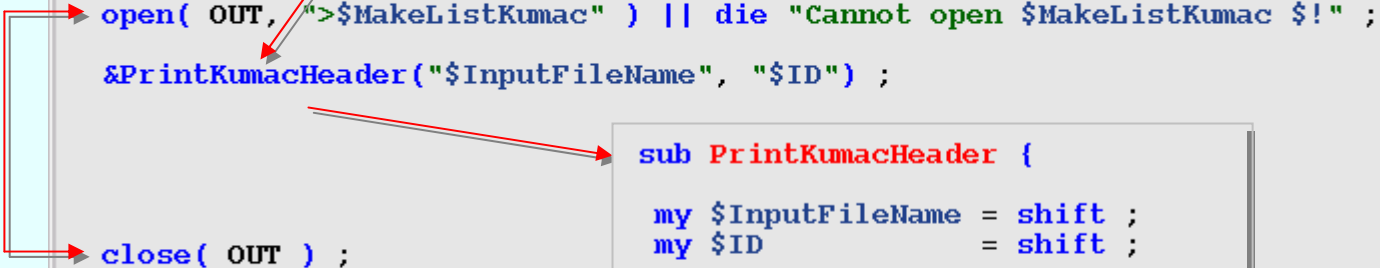
    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;
    &PrintKumacHeader("$InputFileName", "$ID") ;

    close( OUT ) ;
}

sub PrintKumacHeader {
    my $InputFileName = shift ;
    my $ID             = shift ;

    print OUT <<End_Of_Text ;
Macro TempFile
        H/File 1 $InputFileName
        H/Print $ID
End_Of_Text
}
```

Chiamiamo una funzione che scriva le istruzioni nel file KUIP appena aperto (in seguito aggiungeremo altre istruzioni a questo stesso file)



```
sub PrintAdditionalStuff {  
    print OUT <<End_Of_Text ;  
  
    BinsX = \ $HINFO($ID, 'XBINS')  
  
    V/Cre Function([BinsX])  
    V/Cre Contents([BinsX])  
    V/Cre ErrorBar([BinsX])  
  
    H/Get/Func $ID Function  
    H/Get/Cont $ID Contents  
    H/Get/Erro $ID ErrorBar  
  
    V/Pri Function  
    V/Pri Contents  
    V/Pri ErrorBar  
}
```

The screenshot shows a Perl script editor with a menu bar (Preferences, Shell, Macro, Windows, Help) and a title bar (Specialistico/ShowHistogramsFull.pl 4239 bytes). The main window contains Perl code. A sub-routine call `&PrintAdditionalStuff("$ID")` is highlighted with a red arrow pointing to the sub-routine definition in the left-hand code block. Below the call, the code includes `close(OUT) ;`. A terminal window titled "Corso Specialistico di PERL per la CNTC" is open, displaying the output of the script: a list of function values for bins 1 through 24, followed by the prompt "PAW >".

Per i soli istogrammi monodimensionali, aggiungiamo alle istruzioni per PAW quelle che ci permettono di acquisire in vettori i contenuti degli istogrammi, assieme a errori associati alle entries ed eventualmente i valori della fit function presente.



```

sub PrintAdditionalStuff {

    print OUT <<End_Of_Text ;

        BinsX = \$HINFO($ID, 'XBINS')

        V/Cre Function([BinsX])
        V/Cre Contents([BinsX])
        V/Cre ErrorBar([BinsX])

        H/Get/Func $ID Function
        H/Get/Cont $ID Contents
        H/Get/Erro $ID ErrorBar

        V/Pri Function }
        V/Pri Contents }
        V/Pri ErrorBar }

End_Of_Text

```

La stampa in output del contenuto dei vettori con i valori dell'istogramma ci permette, tramite l'uso delle regexp, di catturare nomi e valori in opportune hash che poi manipoleremo a nostro piacimento.

```

Corso Specialistico di PERL per la CNTC
FUNCTION( 1) = ...
FUNCTION(17) = 0
FUNCTION(18) = 245,625
FUNCTION(19) = 208,906
FUNCTION(20) = 175,269
FUNCTION(21) = 147,388
FUNCTION(22) = 126,117
FUNCTION(23) = 110,922
FUNCTION(24) = 100,526
...
CONTENTS( 1) = 188
CONTENTS( 2) = 164
CONTENTS( 3) = 159
CONTENTS( 4) = 178
CONTENTS( 5) = 176
CONTENTS( 6) = 166
CONTENTS( 7) = 181
CONTENTS( 8) = 204
CONTENTS( 9) = 209
CONTENTS(10) = 243
CONTENTS(11) = 249
CONTENTS(12) = 273
CONTENTS(13) = 307
CONTENTS(14) = 292
CONTENTS(15) = 331
CONTENTS(16) = 286
...
ERRORBAR( 1) = 13,7113
ERRORBAR( 2) = 12,8062
ERRORBAR( 3) = 12,6095
ERRORBAR( 4) = 13,3417
ERRORBAR( 5) = 13,2665
ERRORBAR( 6) = 12,8841
ERRORBAR( 7) = 13,4536
ERRORBAR( 8) = 14,2829
ERRORBAR( 9) = 14,4568
ERRORBAR(10) = 15,5885
...

```

```

open( CMD, "paw -b $MakeListKumac | " ) ;
while( <CMD> ) {
  if (      m/Function\((\s*\d+)\) = (.*$)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Function{$X} = $Y ;
  } elsif ( m/Contents\((\s*\d+)\) = (.*$)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Contents{$X} = $Y ;
  } elsif ( m/ErrorBar\((\s*\d+)\) = (.*$)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $ErrorBar{$X} = $Y ;
  } else {
    print ;
  }
}
close( CMD ) ;

```

```
m/Function\((\s*\d+)\) = (.*$)/i
```

```

FUNCTION( 1) ...
FUNCTION(17) = 0
FUNCTION(18) = 245,625
FUNCTION(19) = 208,906
FUNCTION(20) = 175,269
FUNCTION(21) = 147,388
FUNCTION(22) = 126,117
FUNCTION(23) = 110,922
FUNCTION(24) = 100,526
PAW >

```

Vediamo come fare
(un deja vú...)

```

open( CMD, "paw -b $MakeListKumac | " ) ;
while( <CMD> ) {
  if (      m/Function\((\s*\d+)\) = (.*$)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Function{$X} = $Y ;
  } elsif ( m/Contents\((\s*\d+)\) = (.*$)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Contents{$X} = $Y ;
  }
}

```



```

open( CMD, "paw -b $MakeListKumac | " ) ;
while( <CMD> ) {
  if (      m/Function\((\s*\d+)\) = (.*)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Function{$X} = $Y ;
  } elsif ( m/Contents\((\s*\d+)\) = (.*)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Contents{$X} = $Y ;
  } elsif ( m/ErrorBar\((\s*\d+)\) = (.*)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $ErrorBar{$X} = $Y ;
  } else {
    print ;
  }
}
close( CMD ) ;

```

\$Function{18} = 245.625

ell Macro Windows Help
nowHistogramsFull.pl 4239 bytes

Corso Specialistico di PERL per la CNTC

```

FUNCTION( 1) ...
FUNCTION(17) = 0
FUNCTION(18) = 245,625
FUNCTION(19) = 208,906
FUNCTION(20) = 175,269
FUNCTION(21) = 147,388
FUNCTION(22) = 126,117
FUNCTION(23) = 110,922
FUNCTION(24) = 100,526
PAW > █

```

Nella locazione identificata dalla stringa "18" della hash Function mettiamo il valore.

```

open( CMD, "paw -b $MakeListKumac | " ) ;
while( <CMD> ) {
  if (      m/Function\((\s*\d+)\) = (.*)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Function{$X} = $Y ;
  } elsif ( m/Contents\((\s*\d+)\) = (.*)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Contents{$X} = $Y ;
  }
}

```



Infine, unicamente per gli istogrammi mono dimensionali, stampiamo il contenuto dei bin, degli errori associati e della eventuale funzione di interpolazione.

```
ShowHistogramsFull.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/ShowHistogramsFull.pl line 185, col 0, 4239 bytes
    $Y = $2 ;
    $Function{$X} = $Y ;
  } elsif ( m/Contents\((\s*\d+)\) = (.*$)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $Contents{$X} = $Y ;
  } elsif ( m/ErrorBar\((\s*\d+)\) = (.*$)/i ) {
    $X = $1 ;
    $Y = $2 ;
    $ErrorBar{$X} = $Y ;
  } else {
    print ;
  }
}
close( CMD ) ;

if ( defined $mono{$ID} ) {
  print <<End_Of_Text ;

Fit function for histogram $ID in file $InputFileName

  Bin          Content          Fit function
End_Of_Text

  foreach $bin (sort keys %Contents) {
    printf("[%3d] %5d +/- %5d --> %s\n",
      $bin, $Contents{$bin}, $ErrorBar{$bin}, $Function{$bin} ) ;
  }
}
}
```

Usiamo allo scopo l'operatore **defined**, che restituisce un valore non nullo solo se la variabile da esso referenziata esiste nello scope corrente (altra proprietà di introspezione)



```
Corso Specialistico di PERL per la CNTC
<CNTC> ./ShowHistogramsFull.pl histograms.his
```

Vediamo ora il programma in azione



List of histograms found in histograms.his

List of mono-dimensional histograms

```

1000 Full parameters space description
1313 Dalitz Dati
1314 Dalitz Dati
1505 Initial MiniMC Parameters
1506 Fit MiniMC Parameters
1507 Fit Informations
2001 <[PPP] mass (Montecarlo)
3201 NR
3202 <[r](770)
3203 f?!(1275)
3204 f?!(980)
3205 S?!(1475)
3206 <[r](1450)
3207 f?!(400)
3208 f?!(1300)
8000 Input file: master_fitter_genric_ds_ppp.his Output file: ppp_effi_ds.his
    
```

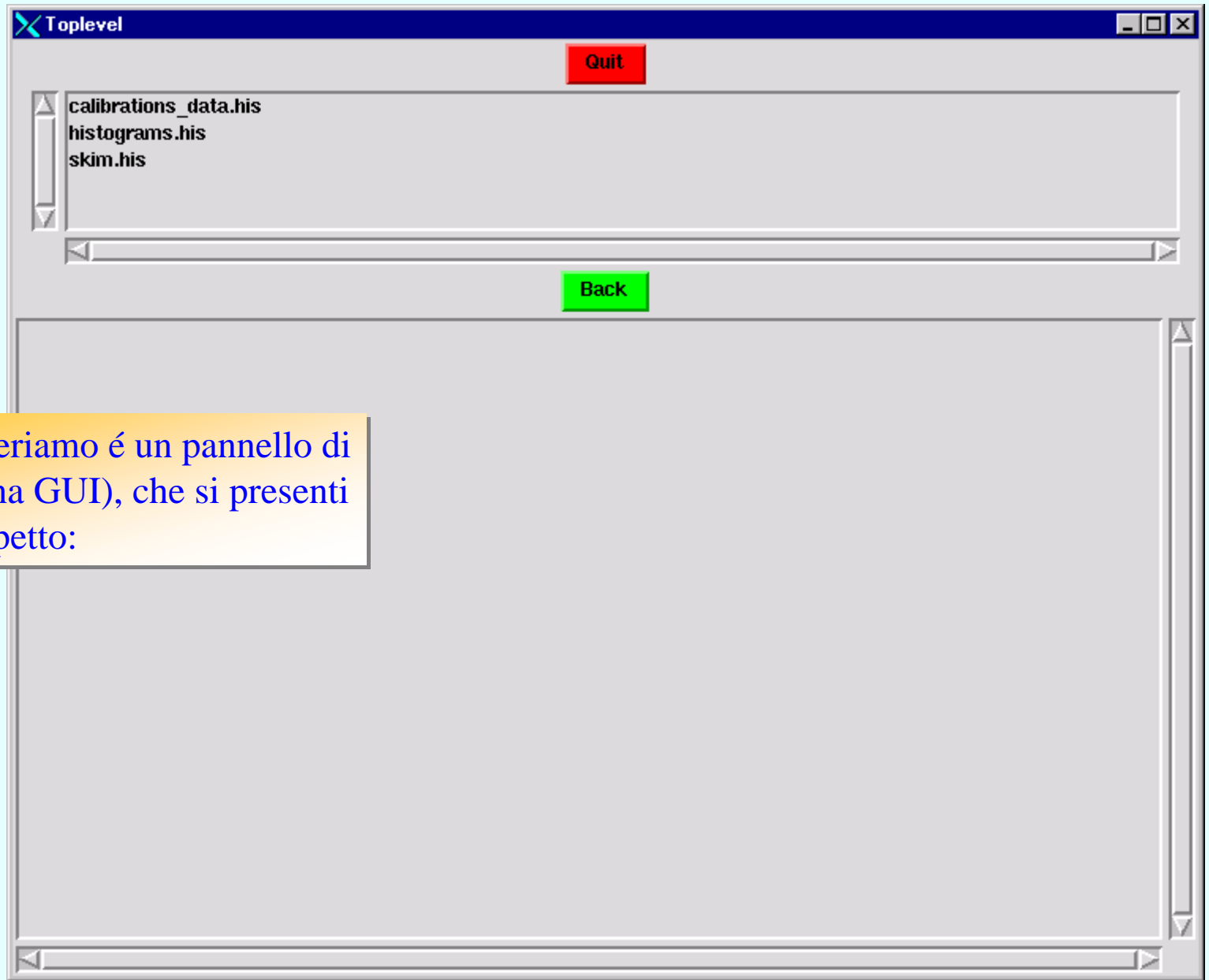
List of bi-dimensional histograms

```

201 NR
202 <[r](770)
203 f?!(1275)
204 f?!(980)
205 S?!(1475)
206 <[r](1450)
207 f?!(400)
208 f?!(1300)
401 1
402 2
1201 Phases NR
1202 Phases <[r](770)
1203 Phases f?!(1275)
1204 Phases f?!(980)
1205 Phases S?!(1475)
1206 Phases <[r](1450)
1207 Phases f?!(400)
1208 Phases f?!(1300)
1501 ds model
1503 ds side bands
1504 Fit Dalitz function (S+B)
1510 Dalitz Dati
1514 Dalitz Effi Check
1515 Dalitz Adaptive Binning
1516 Pure background function from full fit
1517 Fit Dalitz function (Non physical)
1518 Dalitz data (Non physical)
1519 Fit Dalitz function (S)
    
```

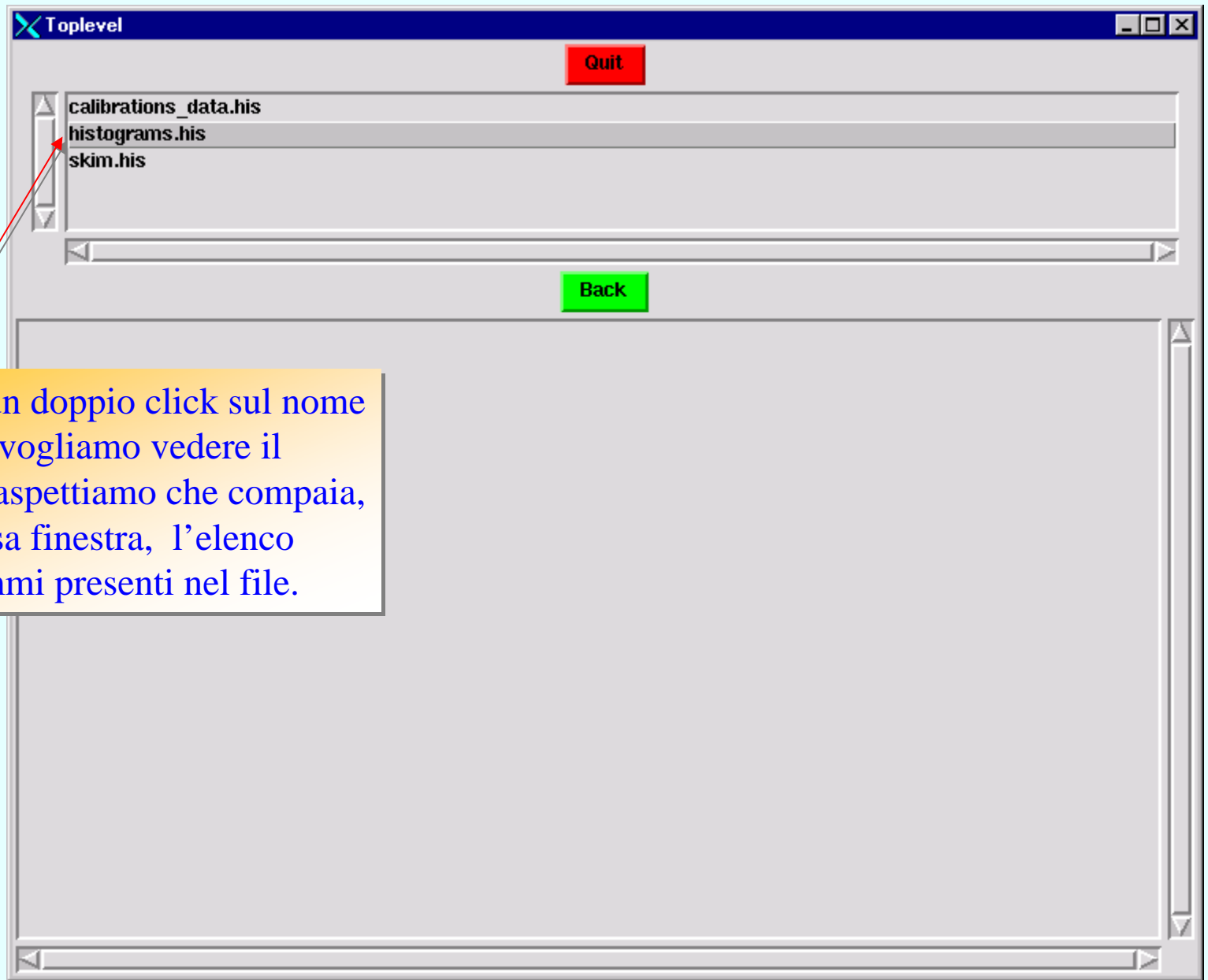
Select an ID: 2001





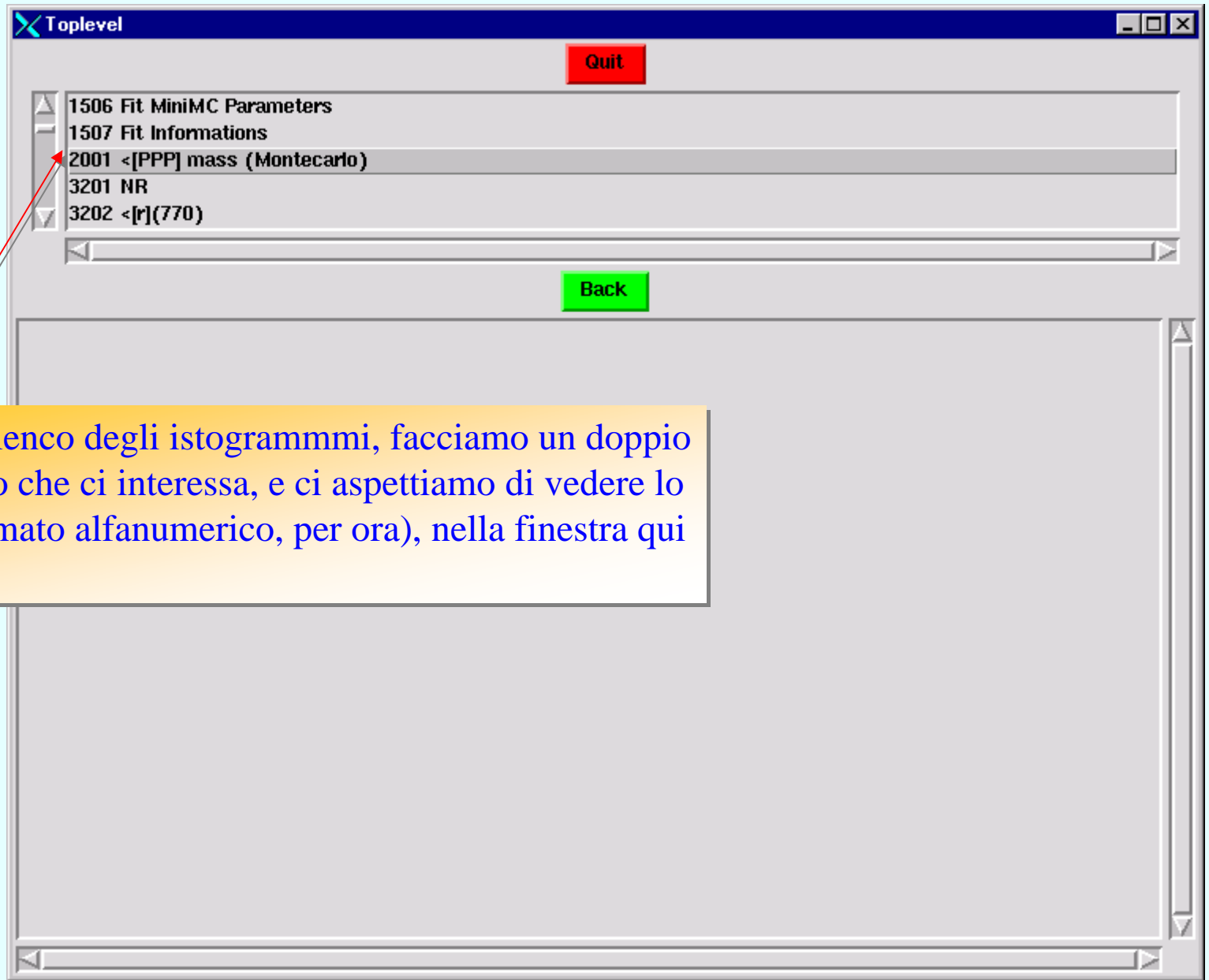
Ciò che desideriamo é un pannello di interfaccia (una GUI), che si presenti con questo aspetto:





Se facciamo un doppio click sul nome del file di cui vogliamo vedere il contenuto, ci aspettiamo che compaia, in questa stessa finestra, l'elenco degli istogrammi presenti nel file.





Comparso l'elenco degli istogrammi, facciamo un doppio click su quello che ci interessa, e ci aspettiamo di vedere lo output (in formato alfanumerico, per ora), nella finestra qui sotto.



Quit

- 1506 Fit MiniMC Parameters
- 1507 Fit Informations
- 2001 <[PPP] mass (Montecarlo)
- 3201 NR
- 3202 <[r](770)

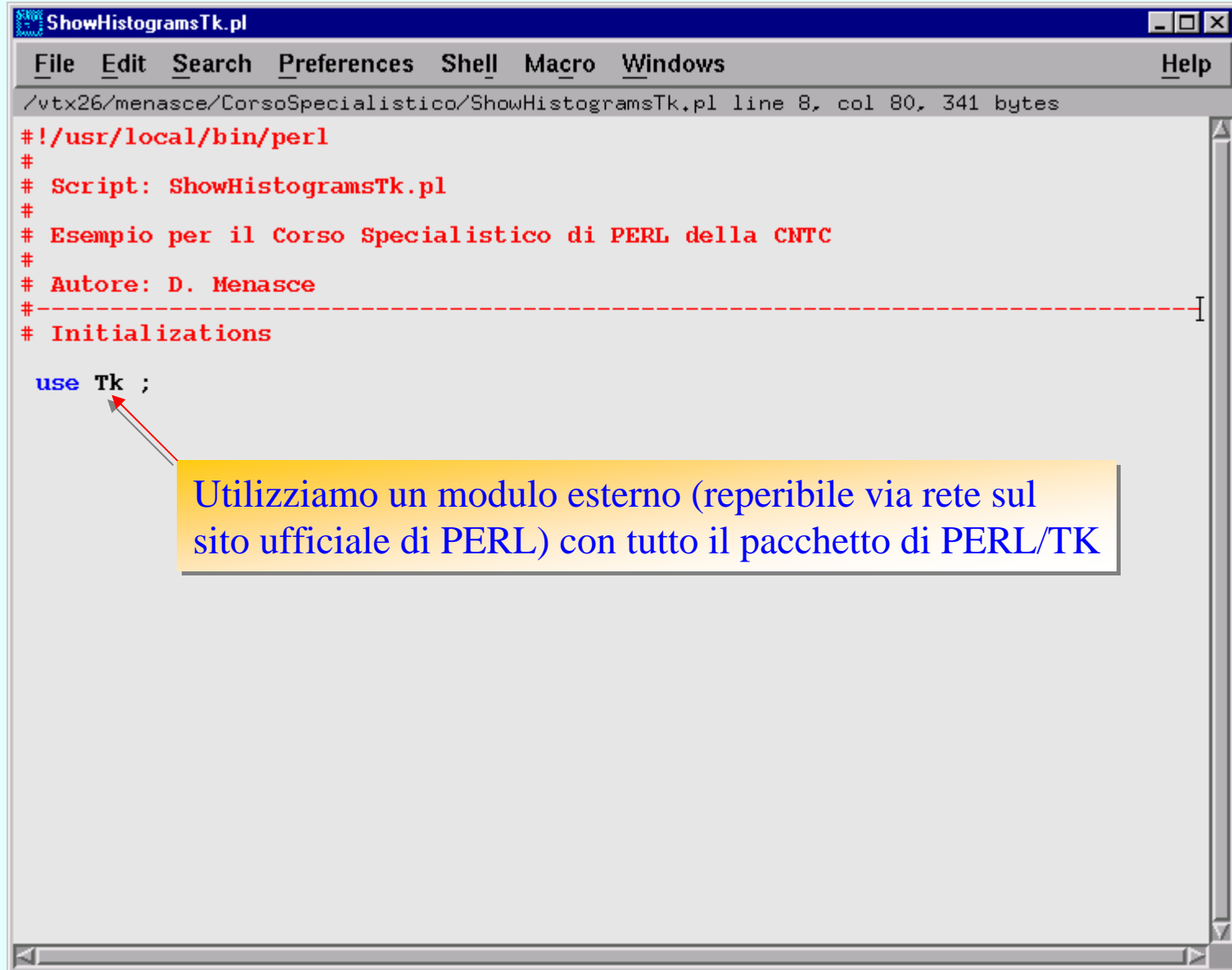
Back

1<[PPP] mass (Montecarlo)

HB00K	ID =	DATE	NO =
410			*
400			I
390			I*
380			II
370			*II
360			II
350			-II
340	-		I I
330	I		I I
320	I		-I I
310	- I -		I I*
300	I-I I		I I-
290	I I-I		* I
280	-I I		I I
270	I I		I I
260	I I-		I I
250	--I *		I I
240	I I		I I
230	I I		I I-
220	I I		I *
210	--I I*		I I
200	I I-		*I I
190	- -I I		I I
180	I -- I I*		I I
170	I- II-I I		I I
160	II-I I		I I

Vediamo come utilizzare PERL per realizzare una GUI di questo tipo





```
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;
```

Utilizziamo un modulo esterno (reperibile via rete sul sito ufficiale di PERL) con tutto il pacchetto di PERL/TK



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 8, col 80, 341 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

```

Creiamo una istanza dell'oggetto **MainWindow** (sintassi *object-oriented*). **\$mw** é una referenza all'oggetto testé creato (per ora soltanto in memoria al processo corrente). L'oggetto **MainWindow** corrisponde ad una finestra (**widget**) vuota che apparirà sullo schermo quando opportunamente richiamata. (Vedremo poi come dotare questa finestra di opportuni strumenti per configurarla come pannello di interfaccia col nostro programma).



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 8, col 80, 341 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

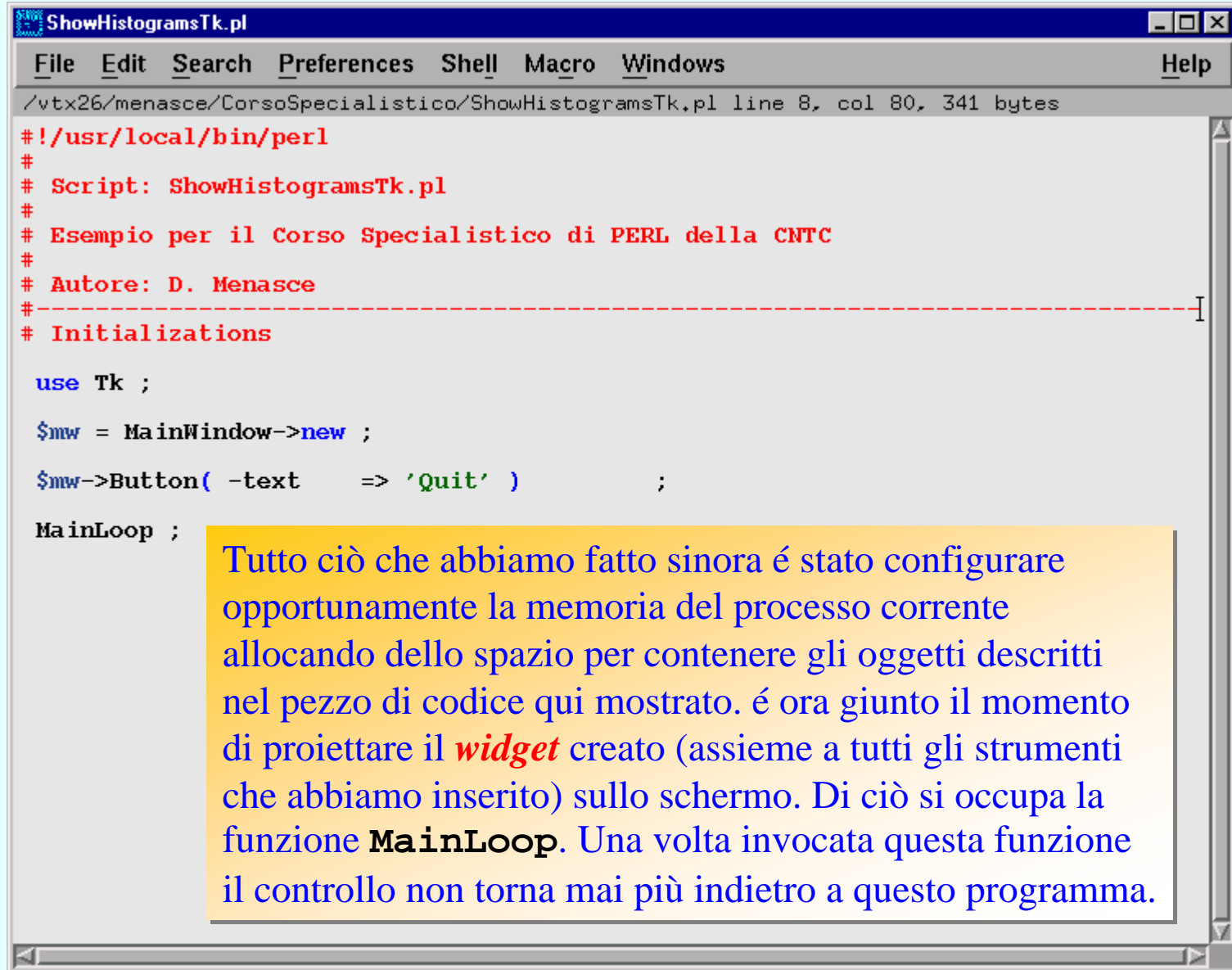
$mw->Button( -text => 'Quit' ) ;

```

Dotiamo il nostro pannello (per ora ancora vuoto) di un bottone premendo il quale il programma vada a termine, rimuovendo il pannello dallo schermo.

Invochiamo il metodo **Button**, metodo posseduto dalla classe **MainWindow** e dotata di parametri pubblici (variabili visibili dall'esterno) di configurazione. In questo esempio configuriamo unicamente il parametro **-text** con la stringa associata **'Quit'**





```
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

$mw->Button( -text => 'Quit' ) ;

MainLoop ;
```

Tutto ciò che abbiamo fatto sinora é stato configurare opportunamente la memoria del processo corrente allocando dello spazio per contenere gli oggetti descritti nel pezzo di codice qui mostrato. é ora giunto il momento di proiettare il *widget* creato (assieme a tutti gli strumenti che abbiamo inserito) sullo schermo. Di ciò si occupa la funzione **MainLoop**. Una volta invocata questa funzione il controllo non torna mai più indietro a questo programma.



Tutto ciò che viene prima di **MainLoop** é quindi usato da **Perl/Tk** per la descrizione del pannello creato assieme alla corrispondenza bottoni/azioni. Le azioni, implementate come funzioni, verranno messe in coda al codice

```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 8, col 80, 341 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

$mw->Button( -text => 'Quit' ) ;

MainLoop ;
```

Un'istruzione successiva a **MainLoop** non viene mai raggiunta. Questo a causa del fatto che **MainLoop** proietta sullo schermo tutti gli strumenti che abbiamo creato, assieme alle istruzioni necessarie affinché un input dell'utente (movimento del mouse, pressione di un tasto della tastiera ecc...) venga messo in corrispondenza con un pezzo di programma che fa qualcosa. Dovremo quindi associare al bottone **Quit** anche una funzione che implementi l'operazione **Quit** secondo il nostro desiderio.



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 8, col 80, 341 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

$mw->Button( -text => 'Quit' ) ;

MainLoop ;

```

Se eseguiamo il programma come scritto sinora, non accade nulla. Il motivo é che le diverse componenti da noi finora create sono solo residenti nella memoria del processo: per renderle visibili occorre invocare il metodo **pack** (che gestisce diversi possibili *geometry managers* deputati alla gestione dell'aspetto grafico dei componenti). Il metodo **pack** possiede anch'esso un gran numero di variabili pubbliche configurabili: noi ci accontentiamo dei suoi valori di default.



The screenshot shows a Perl script editor window titled "ShowHistogramsTk.pl". The code in the editor is as follows:

```
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

$mw->Button( -text => 'Quit' )->pack() ;

MainLoop ;
```

A yellow callout box with a red arrow pointing to the `pack()` method in the code says: "Invochiamo il metodo **pack** ed eseguiamo il programma:".

Below the code, two graphical windows are shown. The first window, titled "Corso Specialistico di PERL per ...", has a terminal-like interface with the prompt "<CNTC> ./ShowHistogramsTk.pl" and a red cursor. The second window, titled "To...", contains a single button labeled "Quit".

Se proviamo a premere il tasto labellato come **Quit** vedremo che non accade nulla. Dobbiamo associare a quel tasto una funzione che implementi la operazione **Quit**




```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 8, col 80, 372 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

$mw->Button( -text    => 'Quit',
            -command => sub {exit})->pack();

MainLoop ;

```

Modifichiamo un'altra delle variabili pubbliche del metodo **Button**, la variabile **command**, che specifica quale comando deve essere eseguito al premere di questo bottone. Il comando che faremo eseguire sarà l'esecuzione di un sottoprogramma (anonimo) che non fa altro che invocare il comando **exit** di **PERL**.



The screenshot shows a Perl/Tk editor window titled "ShowHistogramsTk.pl". The menu bar includes File, Edit, Search, Preferences, Shell, Macro, Windows, and Help. The status bar indicates the file path and current position: "/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 8, col 80, 410 bytes".

```
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

$mw->Button( -text      => 'Quit',
             -background => 'red',
             -command   => sub {exit}->pack();

MainLoop ;
```

A red arrow points from the `-background => 'red'` line in the code to a yellow callout box:

Vogliamo che il bottone sia **rosso**, per cui modifichiamo anche la variabile **background**.
Se eseguiamo ora il programma e premiamo il tasto, il programma va a termine e la finestrella scompare.

To the right, a small window titled "To..." is shown, containing a red button labeled "Quit".



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 8, col 80, 461 bytes
#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$mw = MainWindow->new ;

$bQuit = $mw->Button( -text      => 'Quit',
                    -background => 'red',
                    -command    => sub {exit}) ;

$bQuit->pack() ;

MainLoop ;

```

Il metodo **pack** può essere invocato separatamente in un secondo tempo (potremmo voler far sparire temporaneamente un bottone dal widget e farlo ricomparire dopo). Per far ciò memorizziamo in **\$bQuit** la referenza al bottone **Quit** ed invochiamo in seguito il metodo **pack** a partire dalla sola referenza.



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes

#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

#-----
# Main body

$mw = MainWindow->new ;

&Quit() ;

MainLoop ;

#=====

```

Per rendere il programma più facilmente leggibile e modulare trasferiamo la creazione del bottone **Quit** in una opportuna funzione.

```

sub Quit {
    $bQuit = $mw->Button( -text      => 'Quit',
                        -background => 'red',
                        -command    => sub {exit}) ;

    $bQuit->pack() ;
}

```



```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes

#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

#-----
# Main body

$mw = MainWindow->new ;

&Quit() ;
&CreateScrolledList() ;

MainLoop ;

#=====
```

Vogliamo ora creare una zona scorrevole (mediante cursori) che dovrà contenere la lista dei file di istogrammi presenti nella corrente directory. Invochiamo la funzione **CreateScrolledList** per fare quanto richiesto



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes

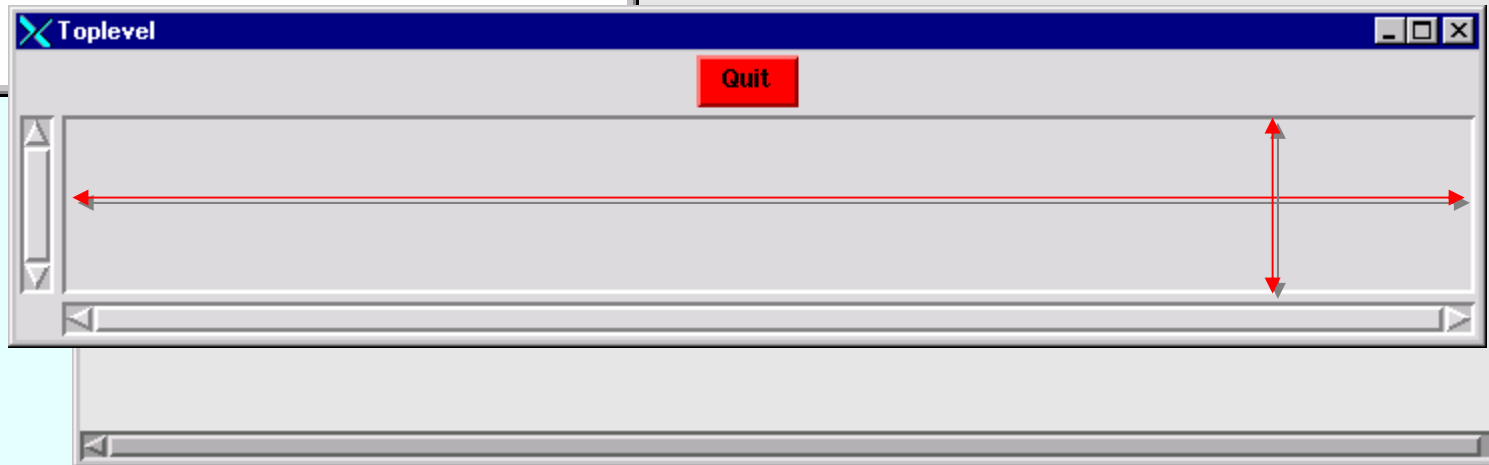
sub CreateScrolledList {
    $ListBoxFrameScroll = $mw->Scrolled("Listbox", -width => 100,
                                              -height => 5 )->pack()
}

```

Corso Specialistico di PERL p

<CNTC> ./ShowHistogramsTk.pl

Invochiamo il metodo **Scrolled** che crea per noi una finestra di questo tipo. Salviamo la referenza all'oggetto creato nella variabile **ListBoxFrameScroll**



```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes

#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

#-----
# Main body

$mw = MainWindow->new ;
@Files = glob "*.his" ;

&Quit() ;
&CreateScrolledList() ;

&ManageScrolledList("SelectFile", @Files) ;

MainLoop ;

#=====
```

Creiamo un vettore con la lista dei file con suffisso **.his**.

Passiamo il vettore come argomento alla procedura **ManageScrolledList**



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes
#-----
sub ManageScrolledList {
    my $Routine = shift ;
    my @Items = @_ ;

    $ListBoxFrameScroll->delete(0, 'end') ;

    {foreach $item (@Items) {
        $ListBoxFrameScroll->insert("end" , "$item " ) ;
    }

    $ListBoxFrameScroll->bind( '<Double-Button-1>', \&$Routine ) ;
}
#-----

```

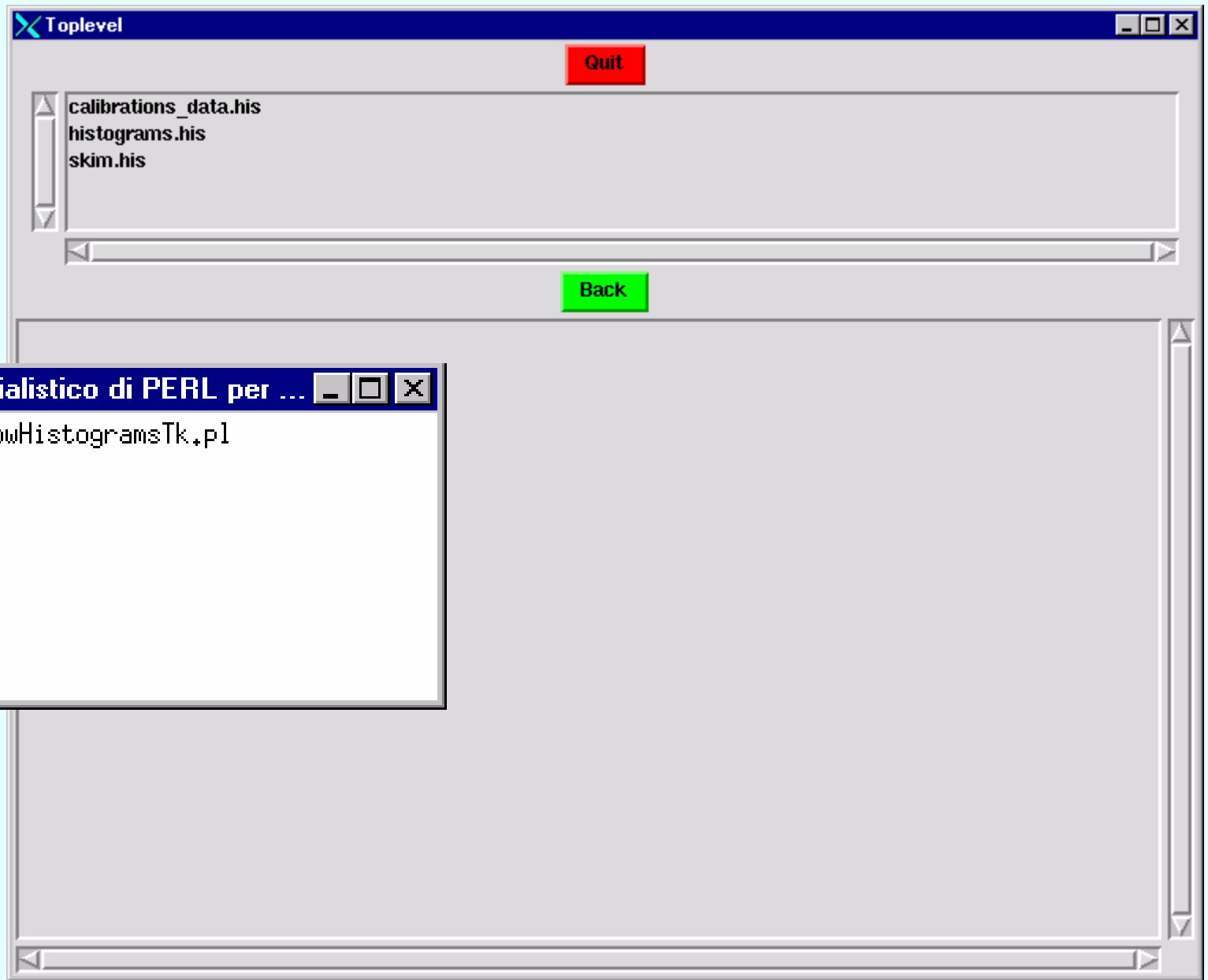
Acquisiamo i due parametri in ingresso rispettivamente in una variabile ed un vettore

Per ogni nome di file in ingresso aggiungiamo una riga all'area scorrevole

Cancelliamo il contenuto dell'area scorrevole

Invochiamo il metodo **bind** per **associare** al contenuto dell'area scorrevole l'esecuzione di una appropriata procedura. Il nome della procedura da chiamare é a sua volta una variabile proveniente dall'esterno (**\$Routine**, che, in questo esempio, vale **SelectFile**). L'associazione é costruita in modo tale per cui un **doppio click sul tasto 1** del mouse (il tasto più a sinistra) ha come effetto l'invocazione immediata della funzione **SelectFile**.





```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes
#=====

sub SelectFile {

    $selection = $ListBoxFrameScroll->curselection();
    $File      = $ListBoxFrameScroll->get($selection);

    &ManageScrolledList("SelectFile", @Files) ;

}

my $Routine = shift ;

#=====

$ListBoxFrameScroll->bind( '<Double-Button-1>', \&$Routine );

```

I metodi `curselection` e `get` (metodi di una `ListBox`) fanno esattamente questo. Al doppio click, la variabile `$File` conterrà la stringa selezionata dall'utente, nel nostro caso il nome di un file.

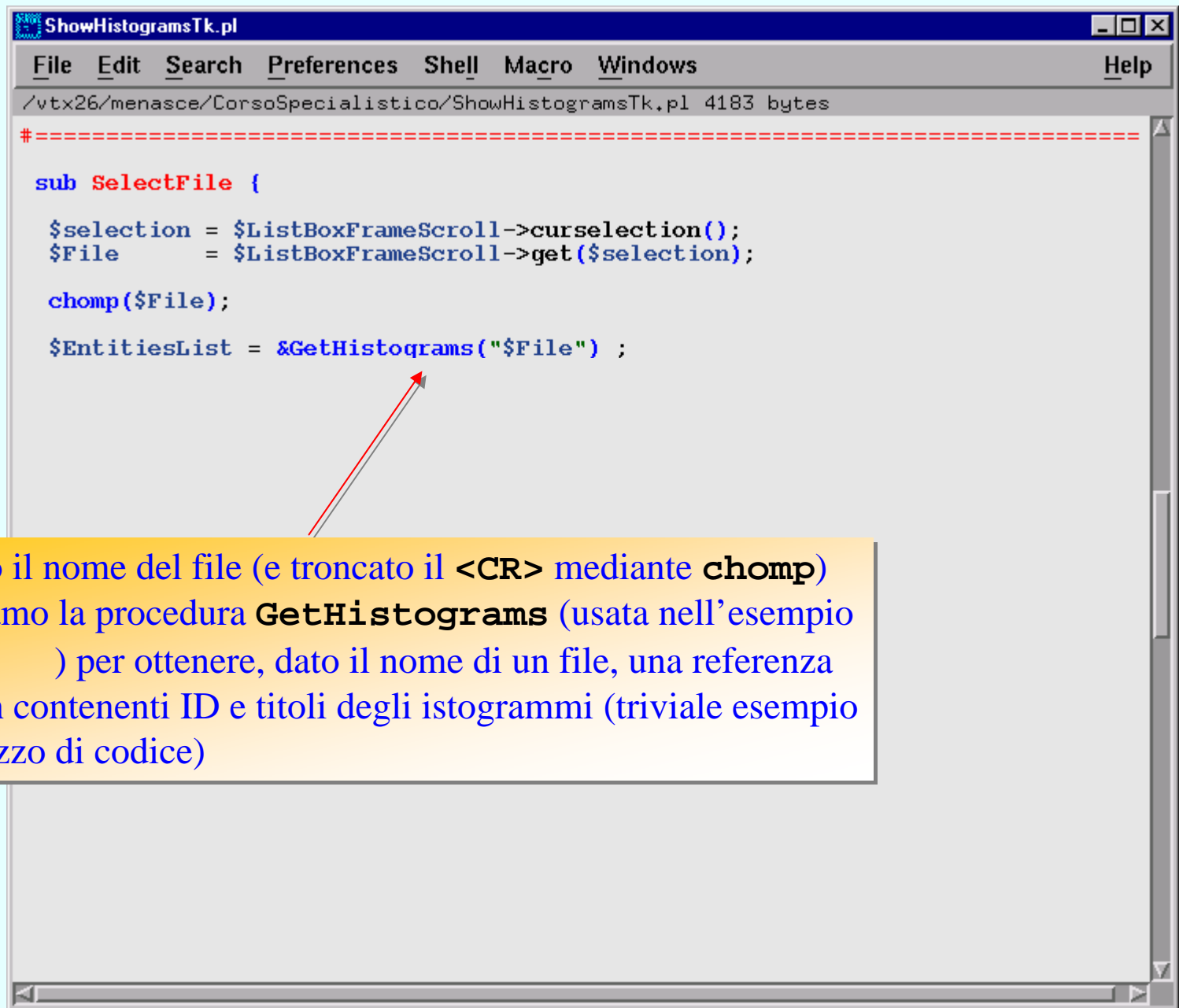
`&ManageScrolledList("SelectFile", @Files) ;`

`my $Routine = shift ;`

`$ListBoxFrameScroll->bind('<Double-Button-1>', \&$Routine);`

Implementiamo ora la funzione `SelectFile`: il suo compito é quello di individuare il nome del file selezionato dall'utente mediante un doppio click e di proporre sulla stessa finestra l'elenco degli istogrammi presenti nel file. Qui useremo una forma di ricorsività





```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes
#=====
sub SelectFile {
    $selection = $ListBoxFrameScroll->curselection();
    $File      = $ListBoxFrameScroll->get($selection);
    chomp($File);
    $EntitiesList = &GetHistograms("$File") ;
}
```

Ottenuto il nome del file (e troncato il `<CR>` mediante `chomp`) invochiamo la procedura `GetHistograms` (usata nell'esempio a pagina) per ottenere, dato il nome di un file, una referenza alle hash contenenti ID e titoli degli istogrammi (triviale esempio di riutilizzo di codice)



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4207 bytes

sub GetHistograms {

    my $InputFileName = shift ;
    my $ID             ;
    my $Type           ;
    my $Title          ;
    my $Name           ;
    my @Types          ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac !" ;

    print OUT <<EOT ;
Macro TempFile

        H/File 1 $InputFileName
        H/Lis

EOT

    close( OUT ) ;

    open( CMD, "paw -b $MakeListKumac | " ) ;
    while( <CMD> ) {
        if ( m/\s+(\d+)\s+\((\d+)\)\s+(.*)/ ) {
            $ID      = $1 ;
            $Type    = $2 ;
            $Title   = $3 ;
            $Name    = $Type{$Type} ;
            $$Name{$ID} = $Title ;
            $TypeNames{$Name}++ ;
        }
    }
    close( CMD ) ;

    return \%TypeNames ;
}

```



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes
#=====

sub SelectFile {

    $selection = $ListBoxFrameScroll->curselection();
    $File      = $ListBoxFrameScroll->get($selection);

    chomp($File);

    $EntitiesList = &GetHistograms("$File") ;

    @HistogramsID = () ;

    foreach $HisType (keys %$EntitiesList) {
        foreach $ID (sort {$a <=> $b} keys %$HisType ) {
            push @HistogramsID, "$ID $$HisType{$ID}" ;
        }
    }

}

```

Abbiamo già visto come si può ciclare sulle hash per riottenere, in ordine alfabetico, gli ID degli istogrammi, prima quelli mono dimensionali e poi quelli bi dimensionali. Accumuliamo nel vettore `@HistogramsID` coppie del tipo “**ID Titolo**”: useremo poi questo vettore per proporre gli istogrammi trovati nella finestra scorrevole.



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes
#=====

sub SelectFile {

    $selection = $ListBoxFrameScroll->curselection();
    $File      = $ListBoxFrameScroll->get($selection);

    chomp($File);

    $EntitiesList = &GetHistograms("$File") ;

    @HistogramsID = () ;

    foreach $HisType (keys %$EntitiesList) {
        foreach $ID (sort {$a <=> $b} keys %$HisType ) {
            push @HistogramsID, "$ID $$HisType{$ID}" ;
        }
    }

    &ManageScrolledList("SelectHistogram", @HistogramsID) ;

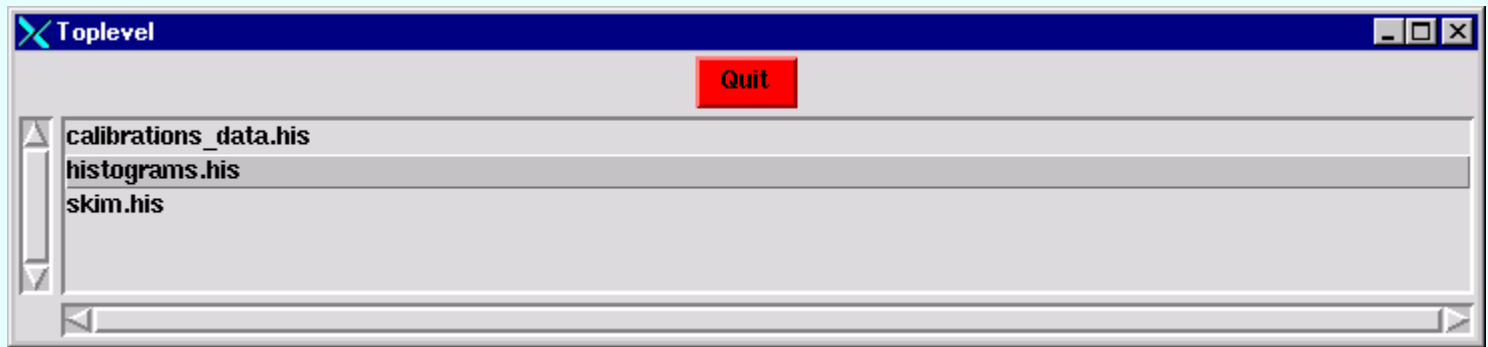
}

#=====

```

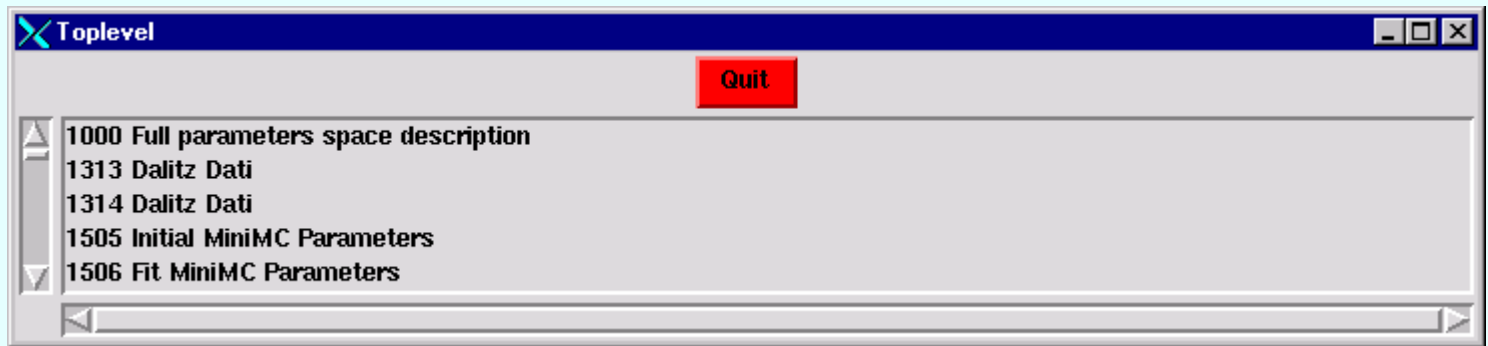
Invochiamo ora la funzione **ManageScrolledList**: ci occorre per associare al doppio click da parte dell'utente su una riga della finestra scorrevole, l'opportuna funzione di rappresentazione dell'istogramma sullo schermo. Questa funzione é chiamata in modo ricorsivo, in quanto l'abbiamo già invocata per elencare i files.





Se facciamo girare ora il programma, otterremo la seguente finestra

Un doppio click su **histograms.his** ed otterremo il seguente risultato:



Ora dobbiamo creare una nuova finestra scorrevole entro la quale rappresentare l'istogramma che l'utente decide di selezionare.



Definiamo il nome di un fonte non proporzionale (ci occorrerà tra breve)

```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4183 bytes

#!/usr/local/bin/perl
#
# Script: ShowHistogramsTk.pl
#
# Esempio per il Corso Specialistico di PERL della CNTC
#
# Autore: D. Menasce
#-----
# Initializations

use Tk ;

$Type{1} = "mono" ;
$Type{2} = "bi" ;
$MakeListKumac = "MakeList.kumac" ;
$FontCourier = "-adobe-courier-medium-r-normal--11-80-100-100-m-60-*-1" ;

#-----
# Main body

$mw = MainWindow->new ;
@Files = glob "*.his" ;

&Quit() ;
&CreateScrolledList() ;
&CreateResultsWidget() ;

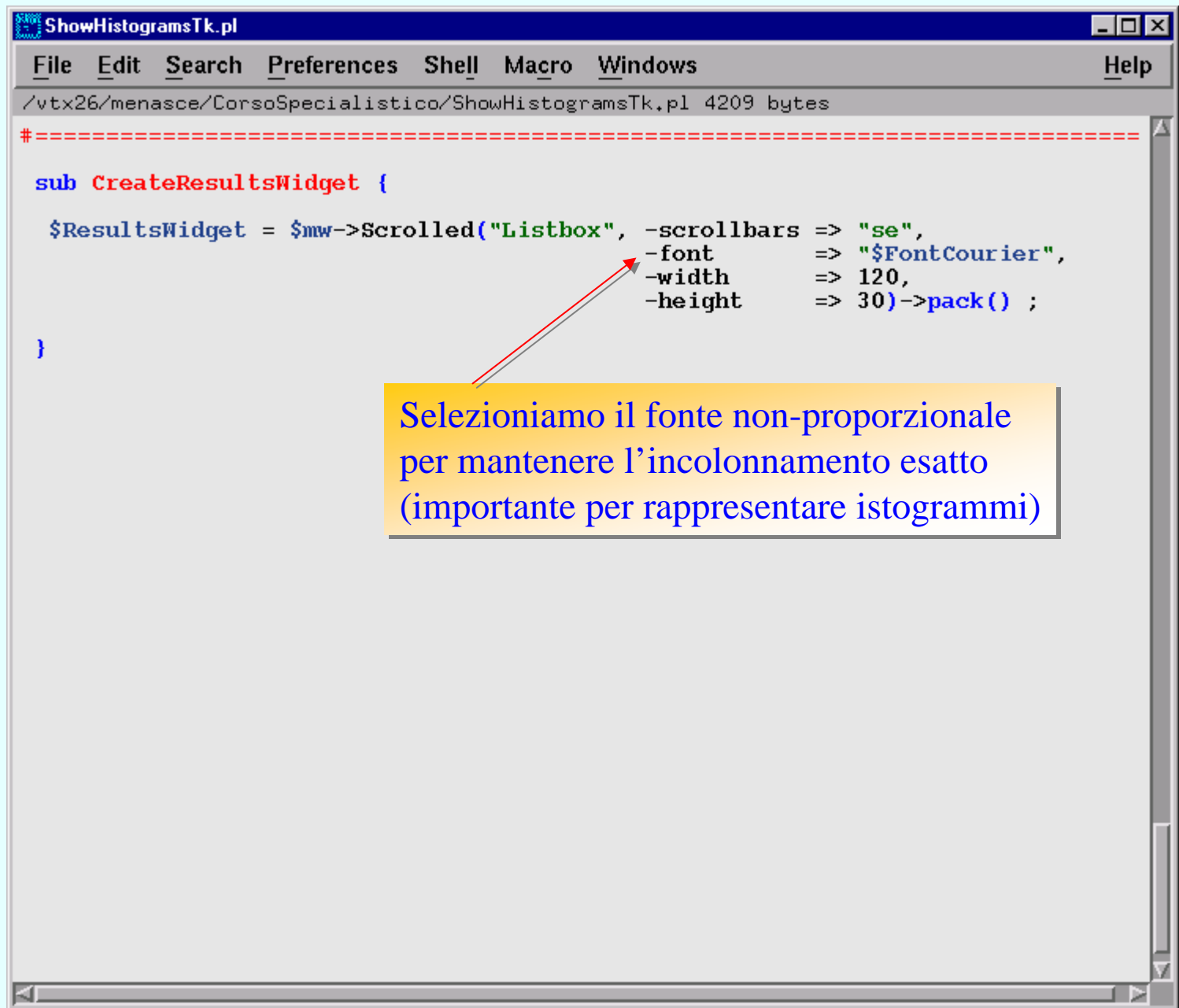
&ManageScrolledList("SelectFile", @Files) ;

MainLoop ;

#=====
```

La nuova finestra scorrevole verrà creata dall' procedura **CreateResultsWidget**



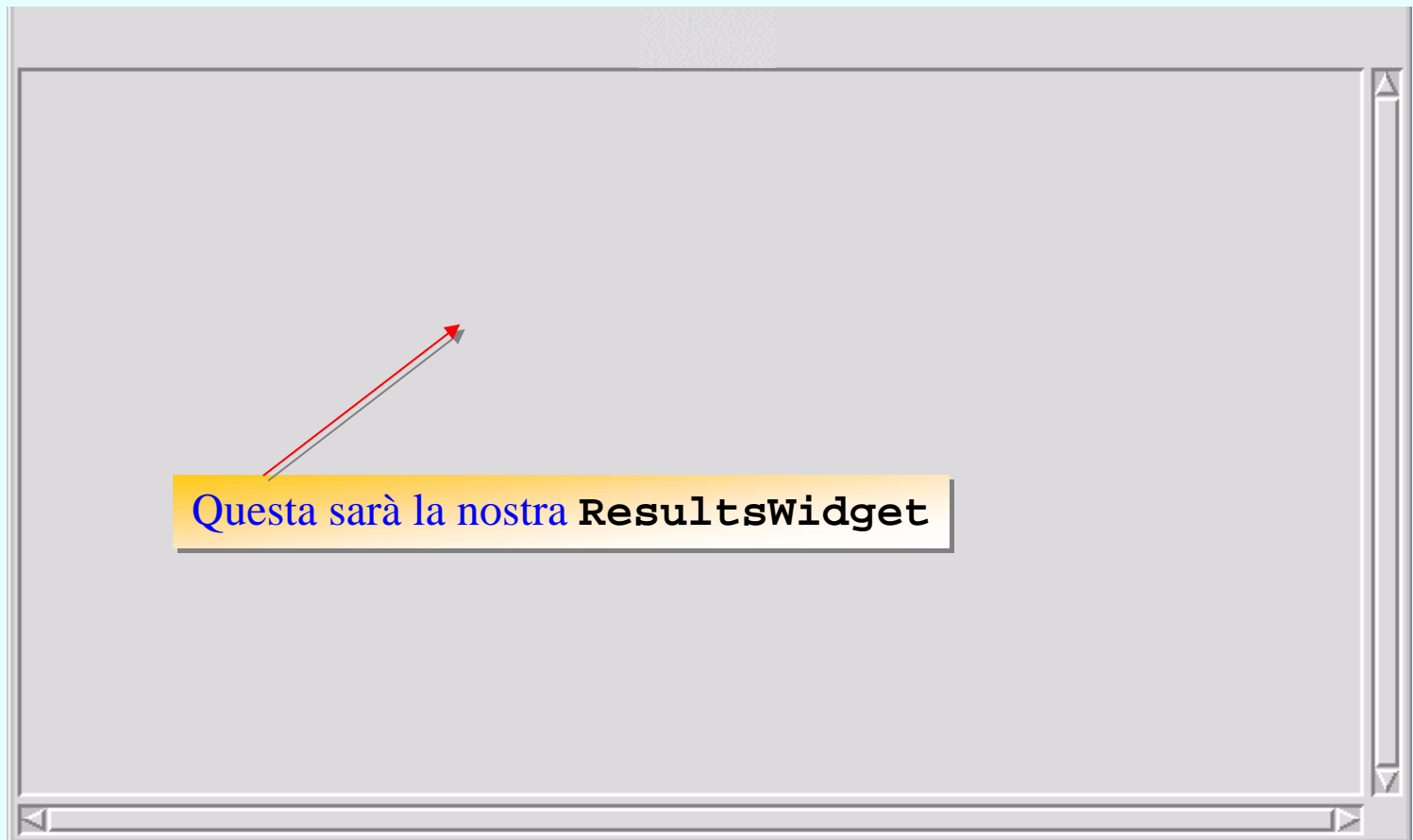
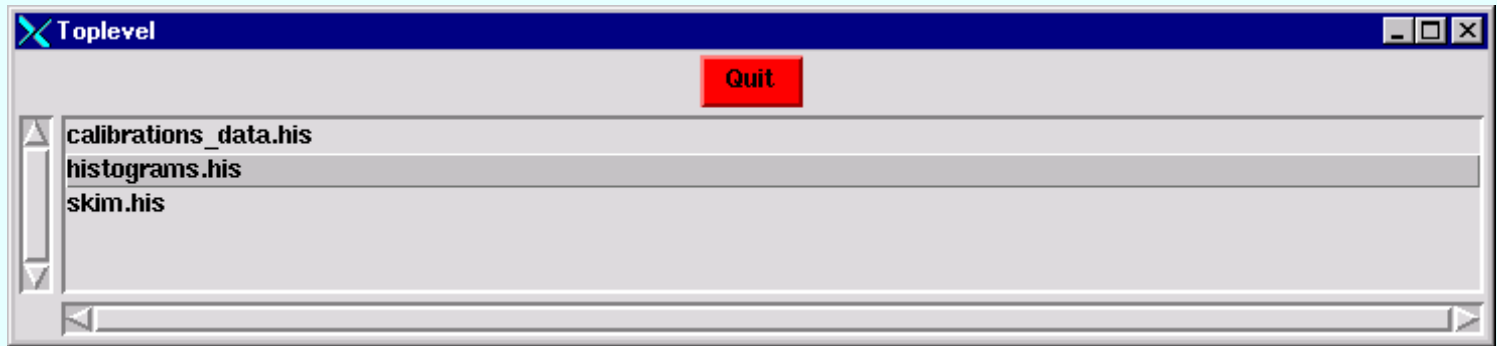


```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl 4209 bytes
#=====
sub CreateResultsWidget {
    $ResultsWidget = $mw->Scrolled("Listbox", -scrollbars => "se",
                                     -font          => "$FontCourier",
                                     -width         => 120,
                                     -height        => 30)->pack() ;
}

```

Selezioniamo il fonte non-proporzionale per mantenere l'incollamento esatto (importante per rappresentare istogrammi)





The image shows a Perl/Tk application window titled "ShowHistogramsTk.pl". The window contains a code editor with the following code:

```
#####  
sub SelectHistogram {  
    $selection = $ListBoxFrameScroll->curselection();  
    $string     = $ListBoxFrameScroll->get($selection);  
    chomp($string);  
}  
#####
```

A yellow callout box with blue text explains the code:

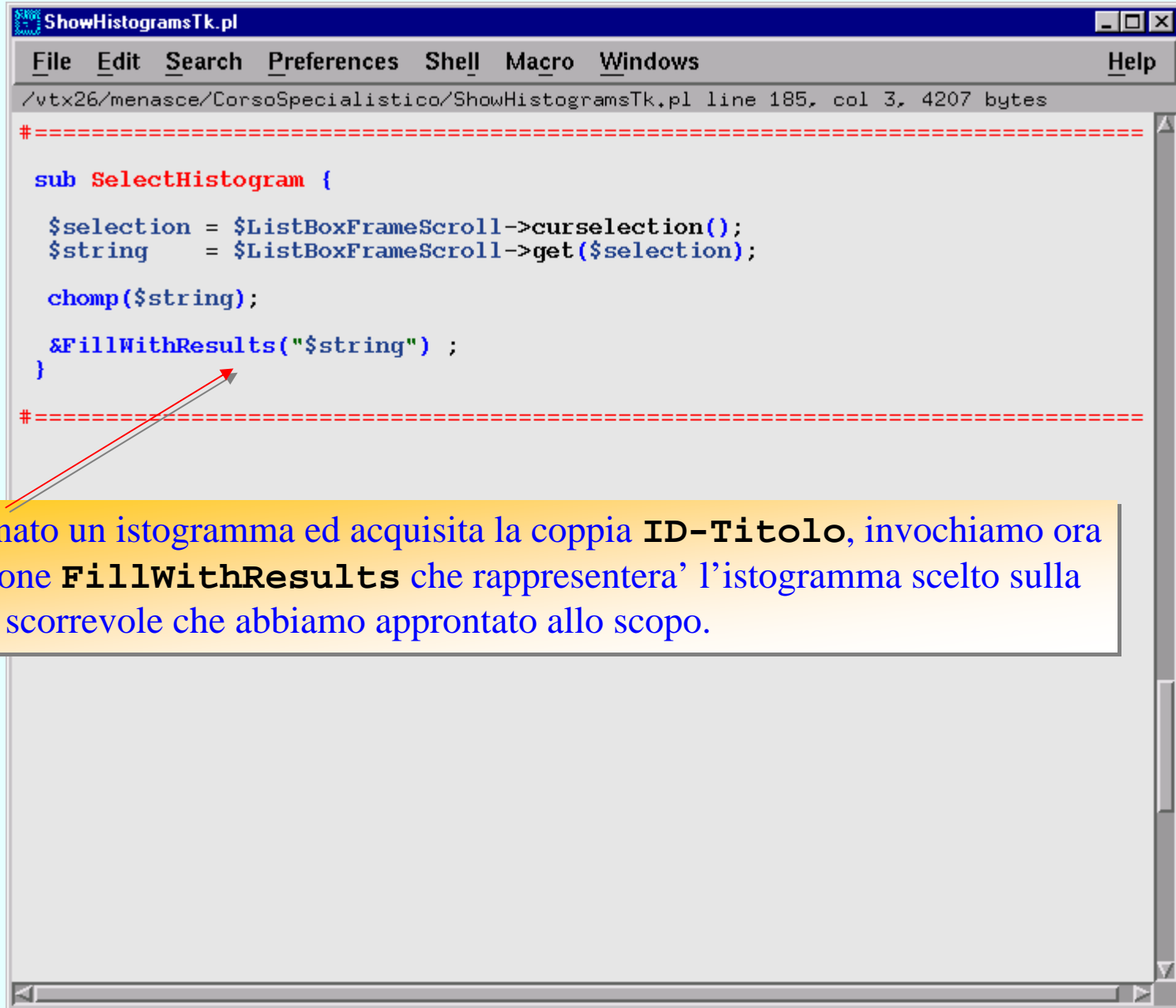
Abbiamo già visto come si acquisisce in una variabile il valore di una riga selezionata dall'utente mediante un doppio-click. In questo caso **string** conterrà la coppia **ID-Titolo**

The "Toplevel" window below shows a list of items:

- 1000 Full parameters space description
- 1313 Dalitz Dati
- 1314 Dalitz Dati
- 1505 Initial MiniMC Parameters
- 1506 Fit MiniMC Parameters

A red "Quit" button is visible in the top right of the Toplevel window. Red arrows indicate the flow of data from the code editor to the Toplevel window.





```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 185, col 3, 4207 bytes
#=====

sub SelectHistogram {

    $selection = $ListBoxFrameScroll->curselection();
    $string     = $ListBoxFrameScroll->get($selection);

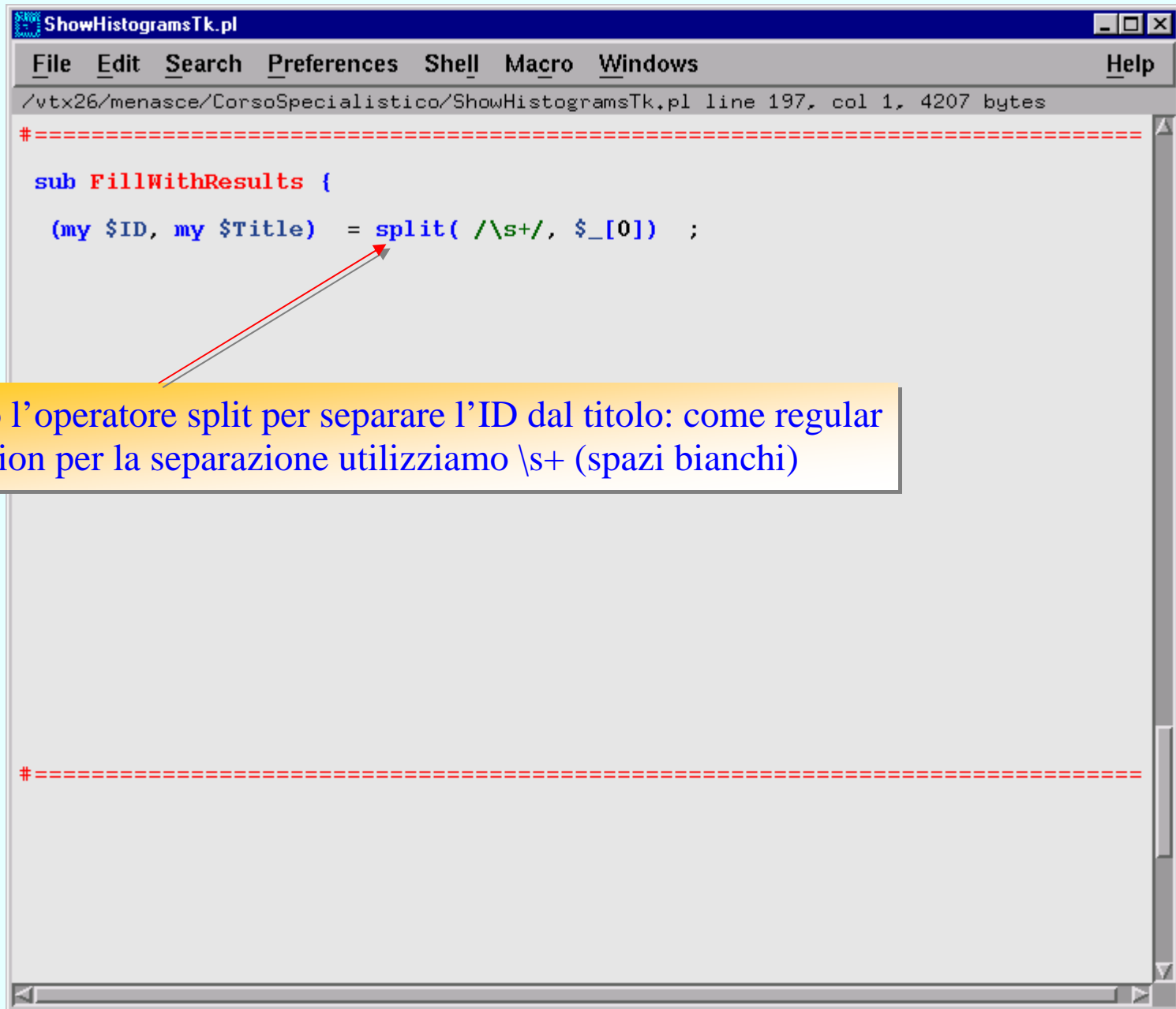
    chomp($string);

    &FillWithResults("$string") ;
}

#=====
```

Selezionato un istogramma ed acquisita la coppia **ID-Titolo**, invochiamo ora la funzione **FillWithResults** che rappresenterà l'istogramma scelto sulla finestra scorrevole che abbiamo approntato allo scopo.





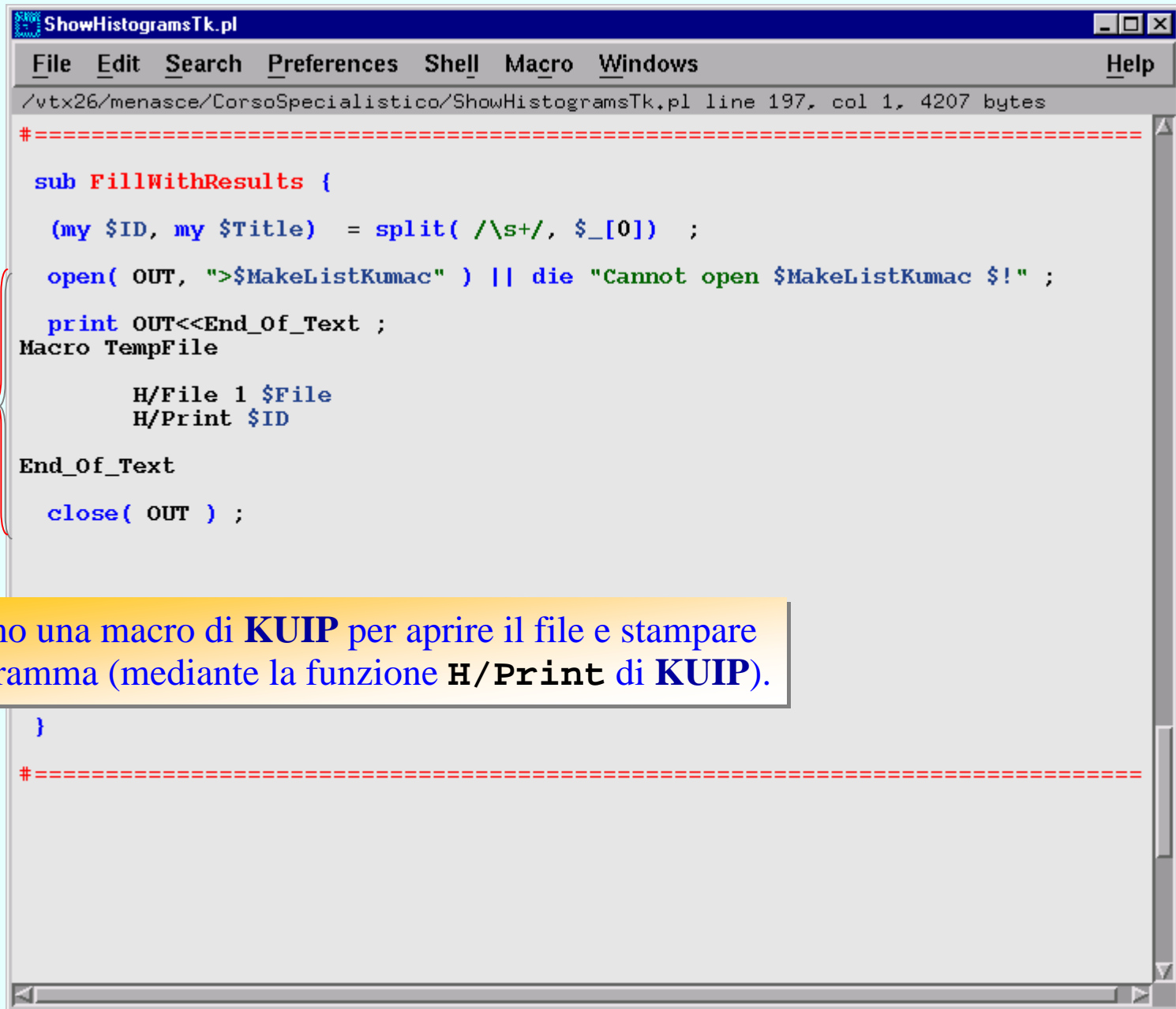
```
ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 197, col 1, 4207 bytes
#=====

sub FillWithResults {
    (my $ID, my $Title) = split( /\s+/, $_[0] ) ;

#=====
```

Usiamo l'operatore split per separare l'ID dal titolo: come regular expression per la separazione utilizziamo \s+ (spazi bianchi)





```
File Edit Search Preferences Shell Macro Windows Help
/vtx26/menasce/CorsoSpecialistico/ShowHistogramsTk.pl line 197, col 1, 4207 bytes
#=====

sub FillWithResults {

    (my $ID, my $Title) = split( /\s+/, $_[0] ) ;

    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;

    print OUT<<End_Of_Text ;
Macro TempFile

        H/File 1 $File
        H/Print $ID

End_Of_Text

    close( OUT ) ;

}

#=====
```

Creiamo una macro di **KUIP** per aprire il file e stampare l'istogramma (mediante la funzione **H/Print** di **KUIP**).



```

ShowHistogramsTk.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/ShowHistogramsTk.pl 4214 bytes
#=====

sub FillWithResults {
    (my $ID, my $Title) = split( /\s+/, $_[0] ) ;
    open( OUT, ">$MakeListKumac" ) || die "Cannot open $MakeListKumac $!" ;
    print OUT<<End_Of_Text ;
Macro TempFile
    H/File 1 $File
    H/Print $ID
End_Of_Text
    close( OUT ) ;
    $ResultsWidget->delete(0, 'end');
    open( CMD, "paw -b $MakeListKumac | " ) ;
    while(<CMD>) {
        chomp ;
        $ResultsWidget->insert("end" , "$_");
    }
    close(CMD) ;
}
#=====

```

Cancelliamo tutte le righe della finestra scorrevole

Apriamo un processo figlio per l'esecuzione di PAW in modalita' batch (solito...)

Tutto ciò che proviene dall'esecuzione di PAW viene proiettato sulla finestra scorrevole



Toplevel

Quit

- 1506 Fit MiniMC Parameters
- 1507 Fit Informations
- 2001 <[PPP] mass (Montecarlo)
- 3201 NR
- 3202 <[r](770)

Back

1<[PPP] mass (Montecarlo)

HB00K ID = 2001 DATE 23/01/:1 NO = 1

410			*		
400			I		
390			I*		
380			II		
370			*II		
360			II		
350			-II		
340	-		I I		
330	I		I I		
320	I		-I I		
310	- I -		I I*		
300	I-I I		I I-		
290	I I-I		* I		
280	-I I		I I	-	
270	I I		I I	*	
260	I I-		I I	I	
250	--I *		I I	*I*	
240	I I		I I	-I-	
230	I I		I I-	I I	
220	I I		I *	-I I	
210	--I I*		I I	* I*	
200	I I-		*I I	I I-	
190	- -I I		I I	I I	
180	I -- I I*		I I	I I	
170	I- II-I I		I I	-I I	
160	II-I I		I I	* I-	



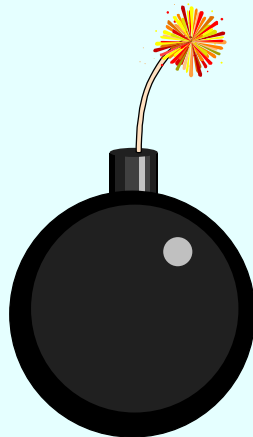
Utilizzo del protocollo CGI connesso a procedure in PERL per la creazione di pagine WEB dinamiche

- Come faccio a creare una home page?
- Come faccio a far girare un mio programma dalla mia home page?

La risposta alla prima domanda é semplice e con un breve esempio daremo un punto di partenza per lo sviluppo di una home page priva di contenuti dinamici. Allo scopo occorre poco più di un text editor e di un browser (più qualche tool di grafica se vogliamo decorare la pagina con fotografie e disegni)

La risposta alla seconda domanda é invece più articolata: occorre conoscere qualche elemento del protocollo CGI, che prescrive come attivare un programma eseguibile su una macchina remota tramite un WEB browser e come passare dei parametri a questo programma.



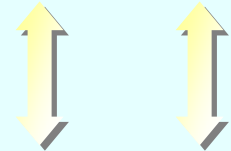
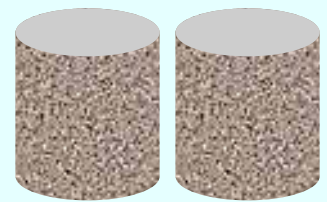


(Gli esempi riportati in questo corso si riferiscono strettamente al modo in cui sono configurate le macchine del CNAF di Bologna. Ogni installazione di un WEB server é infatti potenzialmente differente, per cui i dettagli implementativi dipendono dal particolare server utilizzato).

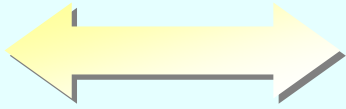




File con pagine HTML su disco



TCP/IP



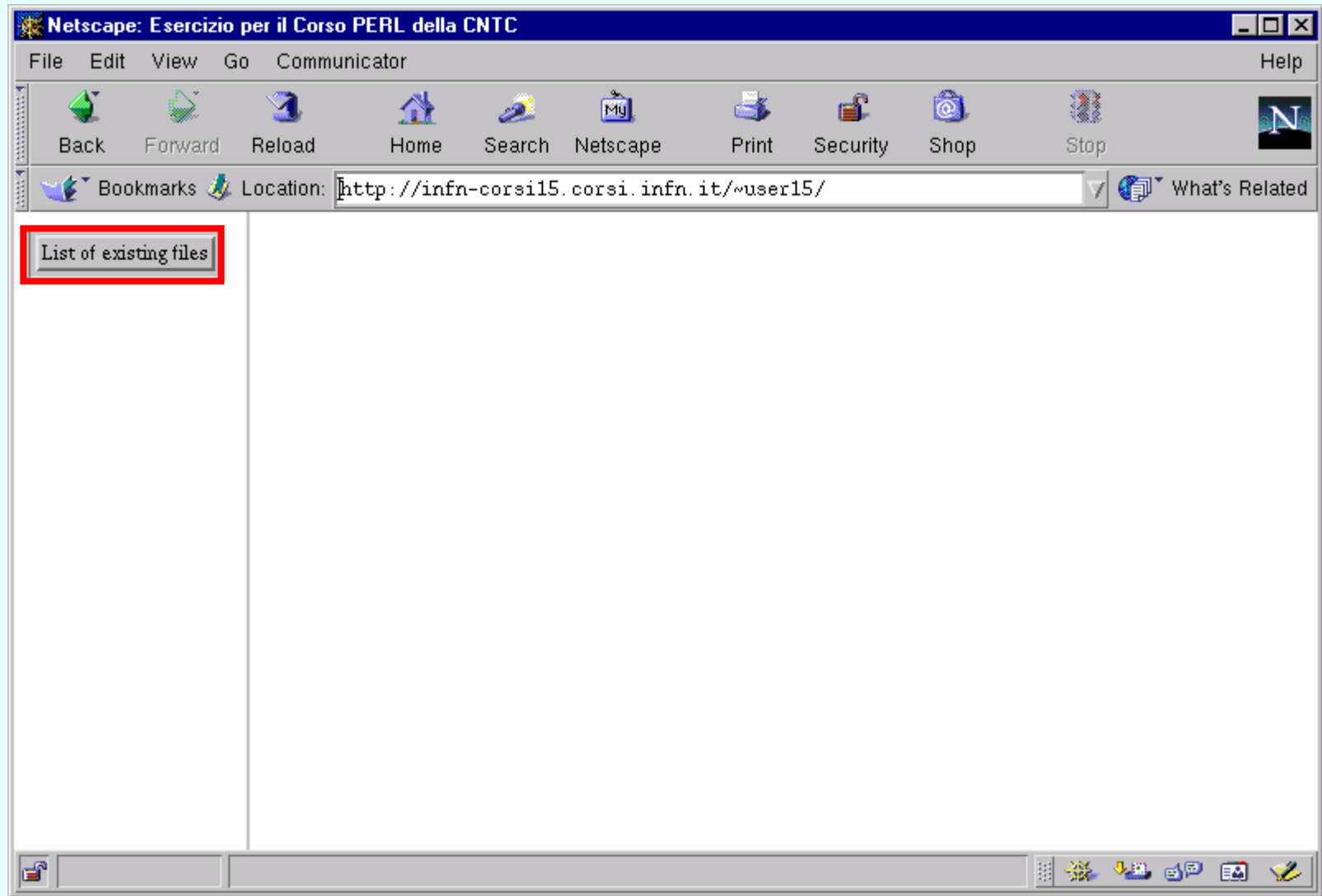
Client

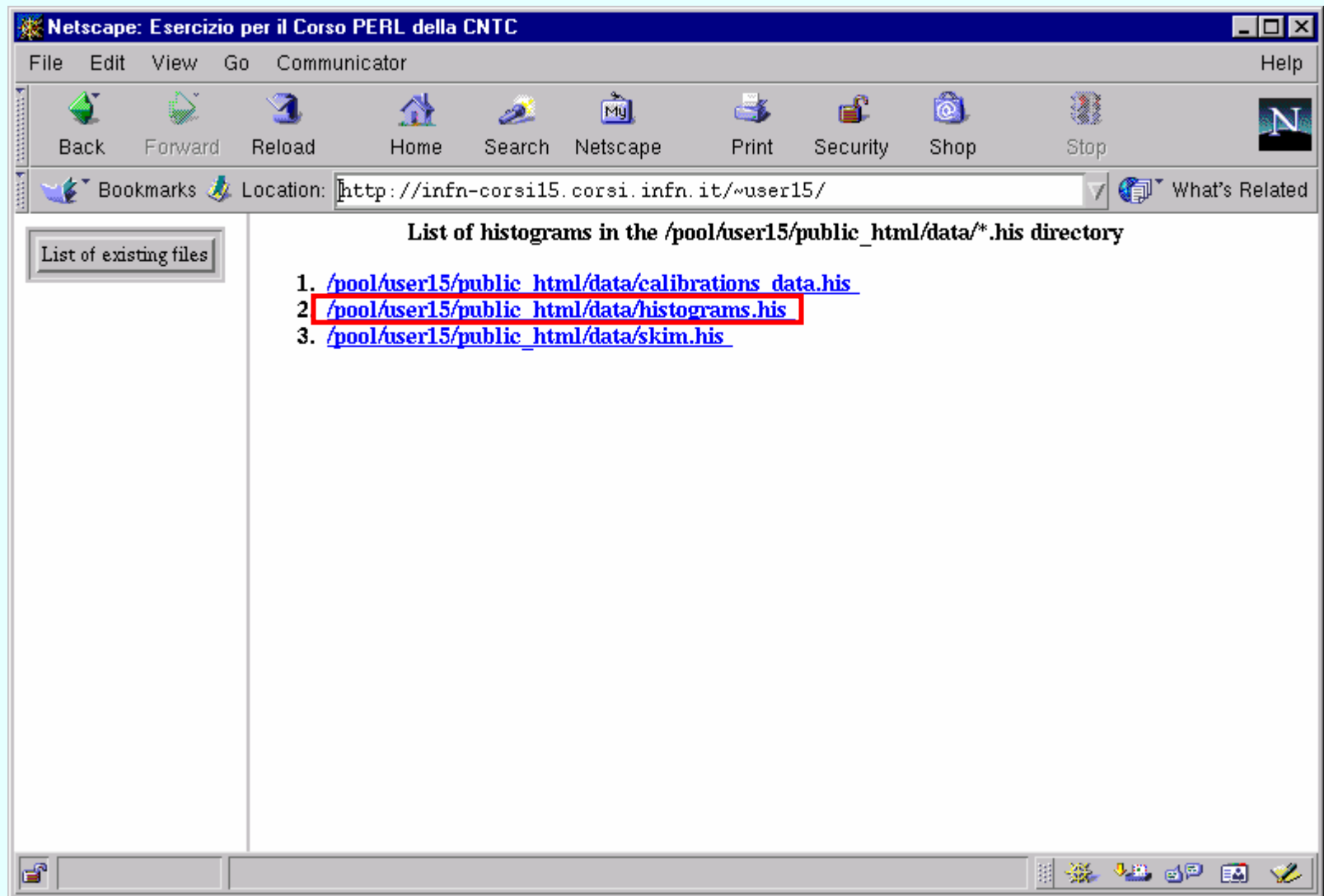
Server

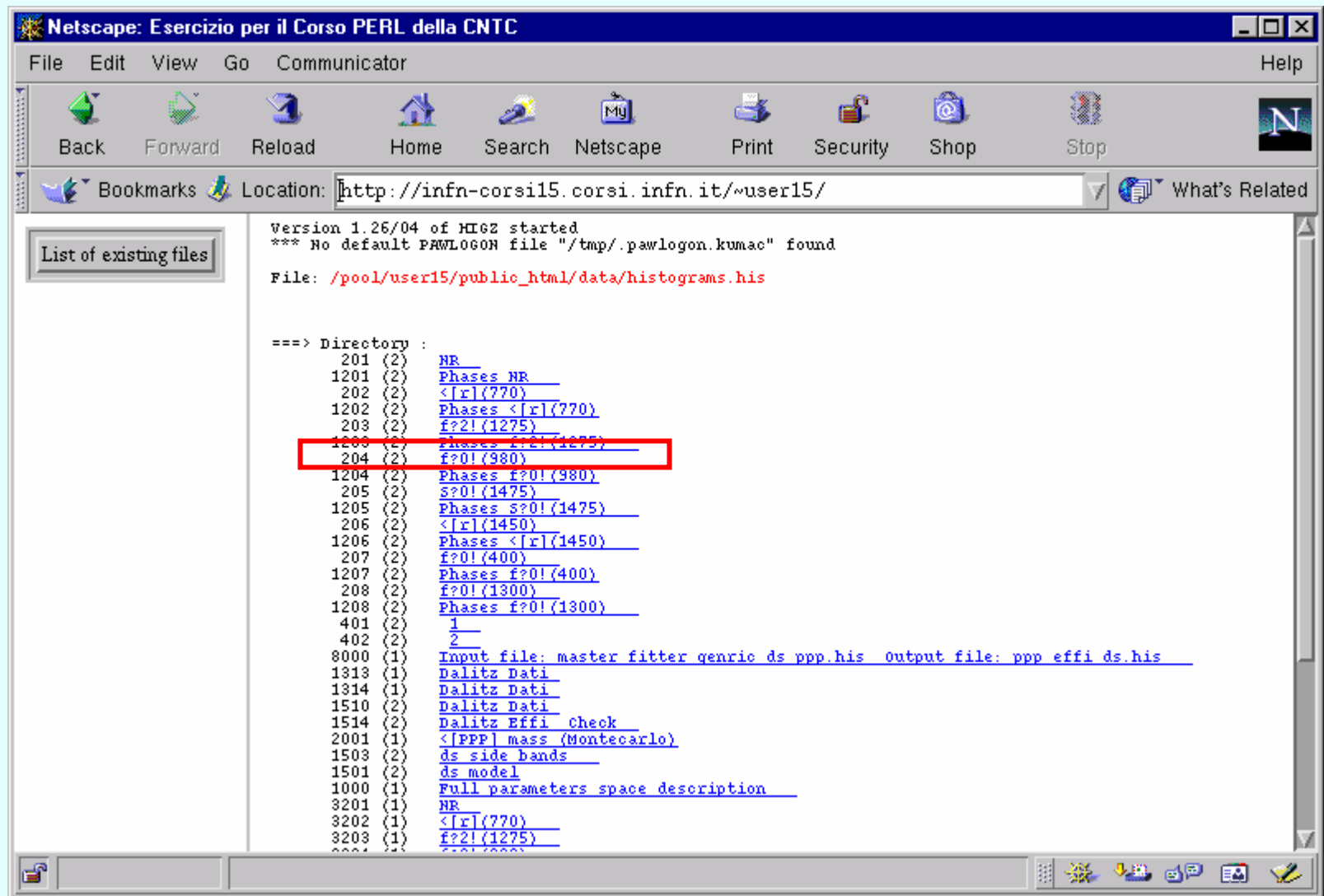
File di configurazione con le autorizzazioni per gli utenti (security)



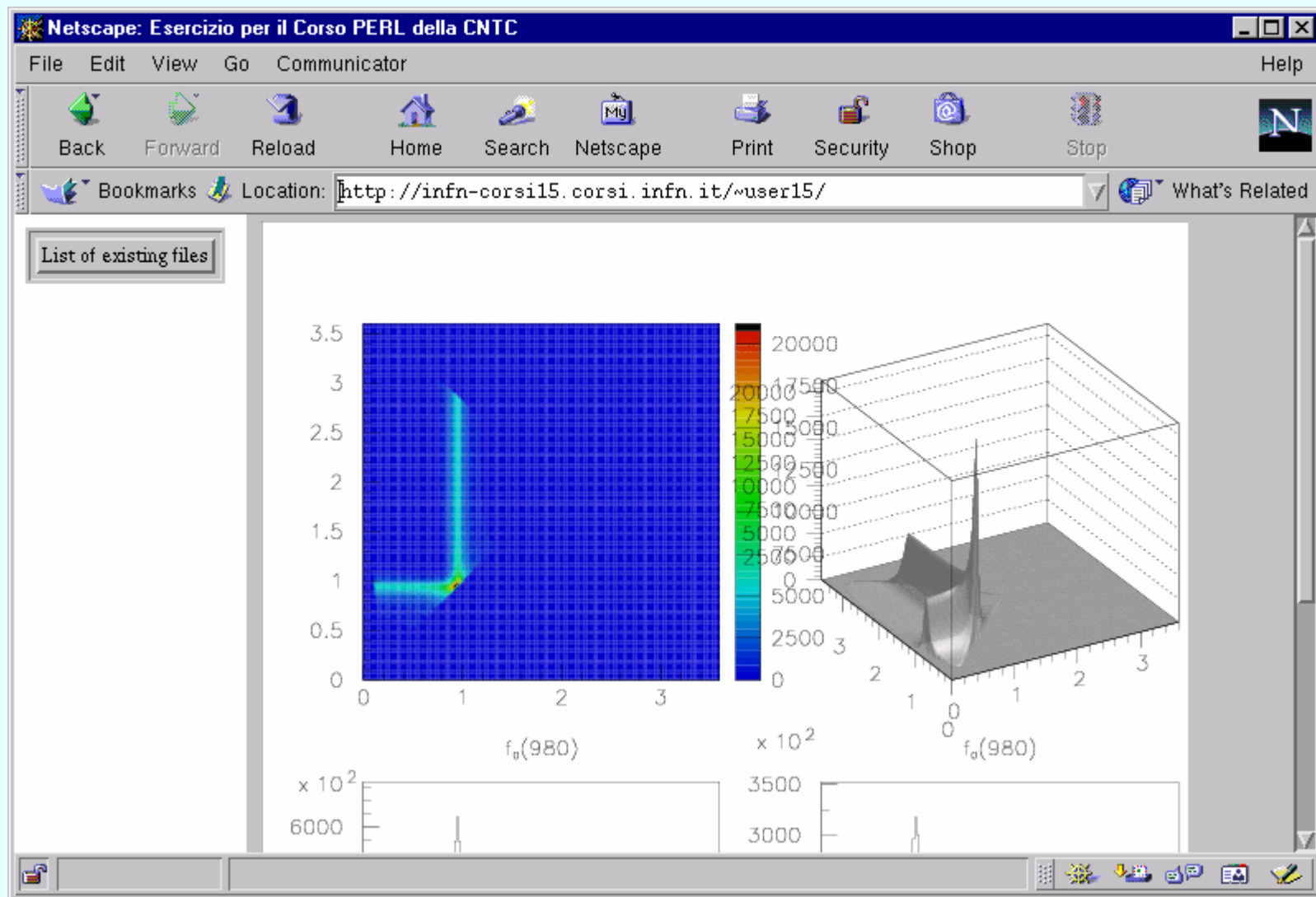
Vogliamo ora realizzare un server di istogrammi, interfacciato alla WEB, che funzioni in modo simile al prototipo realizzato in PERL/Tk. Un oggetto del genere:







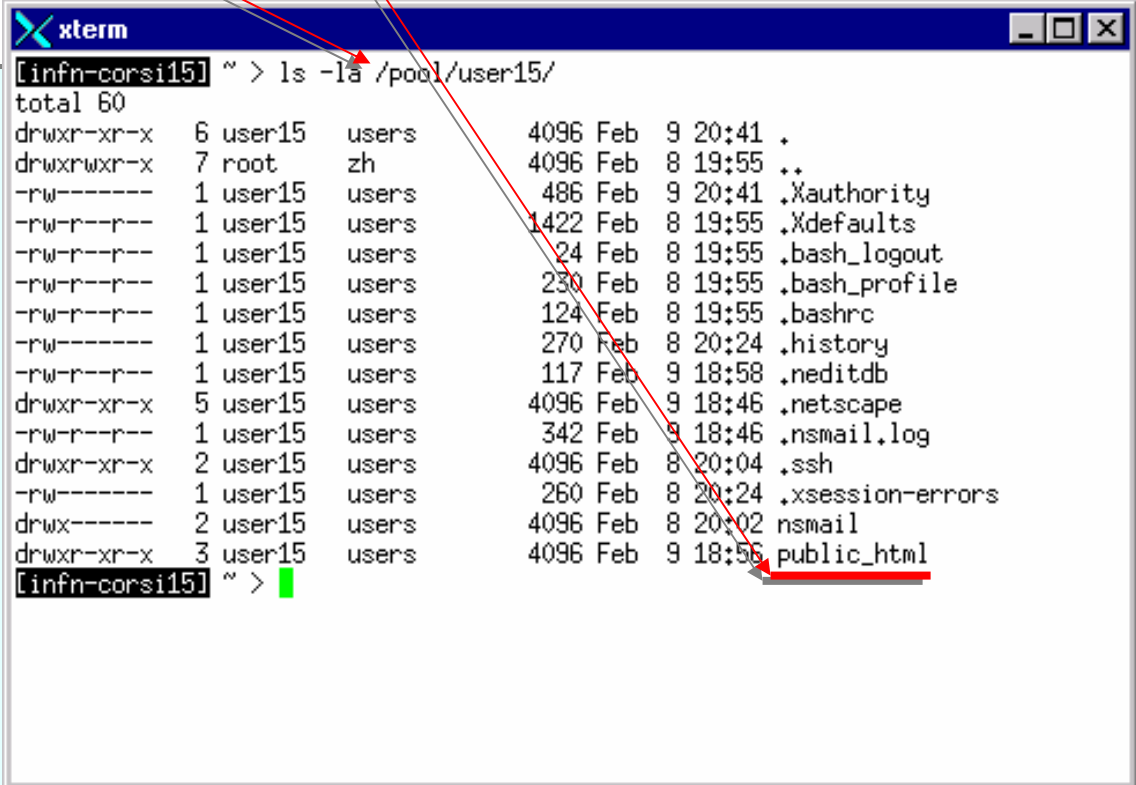
Vediamo da dove iniziare per realizzare qualcosa di così sofisticato



File di configurazione del server httpd (apache)

```
...
UserDir public_html
...
<Directory /pool/user15/public_html>
  Options Indexes SymLinksIfOwnerMatch
  ...
</Directory>
```

/etc/httpd/conf/httpd.conf



```
xterm
[infncorsi15] ~ > ls -la /pool/user15/
total 60
drwxr-xr-x 6 user15 users 4096 Feb 9 20:41 .
drwxrwxr-x 7 root zh 4096 Feb 8 19:55 ..
-rw----- 1 user15 users 486 Feb 9 20:41 .Xauthority
-rw-r--r-- 1 user15 users 1422 Feb 8 19:55 .Xdefaults
-rw-r--r-- 1 user15 users 24 Feb 8 19:55 .bash_logout
-rw-r--r-- 1 user15 users 230 Feb 8 19:55 .bash_profile
-rw-r--r-- 1 user15 users 124 Feb 8 19:55 .bashrc
-rw----- 1 user15 users 270 Feb 8 20:24 .history
-rw-r--r-- 1 user15 users 117 Feb 9 18:58 .netidb
drwxr-xr-x 5 user15 users 4096 Feb 9 18:46 .netscape
-rw-r--r-- 1 user15 users 342 Feb 9 18:46 .nsmail.log
drwxr-xr-x 2 user15 users 4096 Feb 8 20:04 .ssh
-rw----- 1 user15 users 260 Feb 8 20:24 .xsession-errors
drwx----- 2 user15 users 4096 Feb 8 20:02 nsmail
drwxr-xr-x 3 user15 users 4096 Feb 9 18:55 public_html
[infncorsi15] ~ >
```

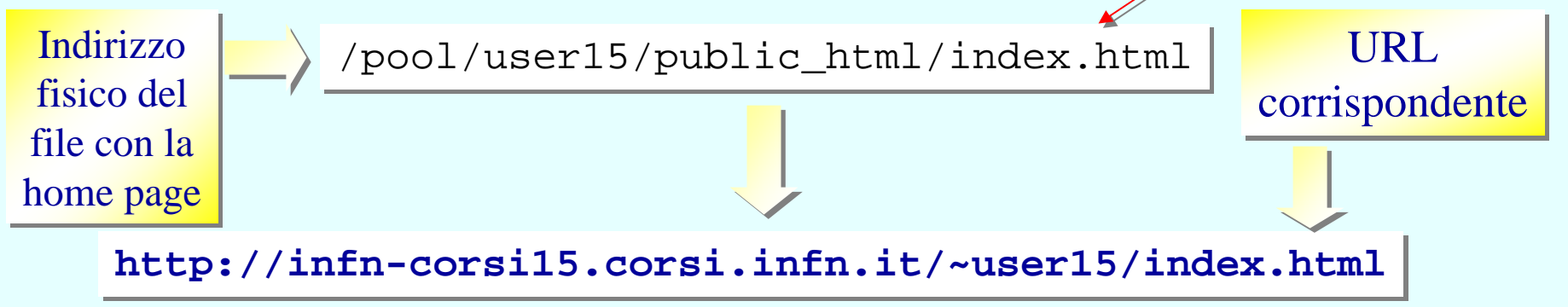


```
xterm
[infncorsi15] ~/public_html > ls -la
total 16
drwxr-xr-x  3 user15  users    4096 Feb  9 20:54 .
drwxr-xr-x  6 user15  users    4096 Feb  9 20:41 ..
drwxr-xr-x  2 user15  users    4096 Feb  9 18:59 cgi-bin
-rw-r--r--  1 user15  users    161 Feb  9 20:54 index.html
[infncorsi15] ~/public_html >
```

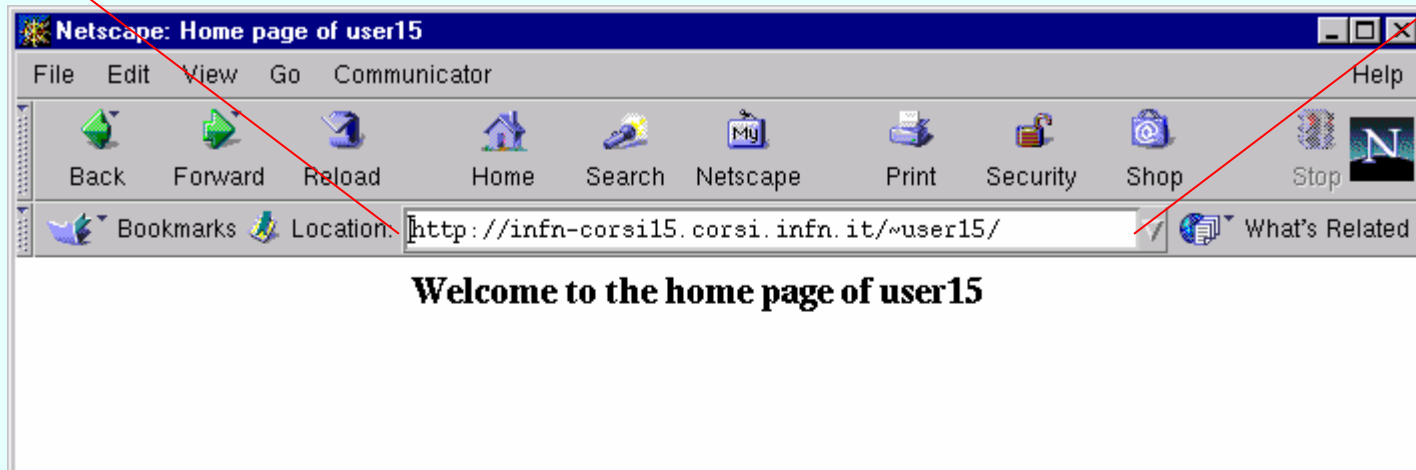
```
index.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/index.html line 15, col 0, 161 bytes
<HTML>
<TITLE>Home page of user15</TITLE>
<Body BgColor="White">

<H2>
<Center>
  Welcome to the home page of <B>user15</B>
</Center>

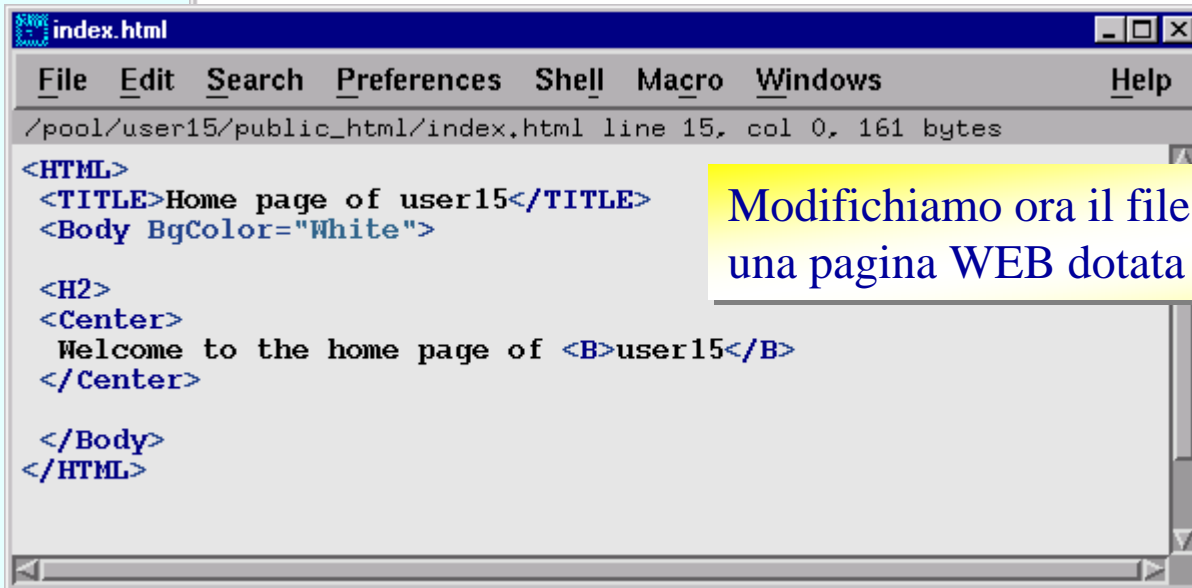
</Body>
</HTML>
```



<http://infncorsi15.corsi.infn.it/~user15/index.html>



Welcome to the home page of user15

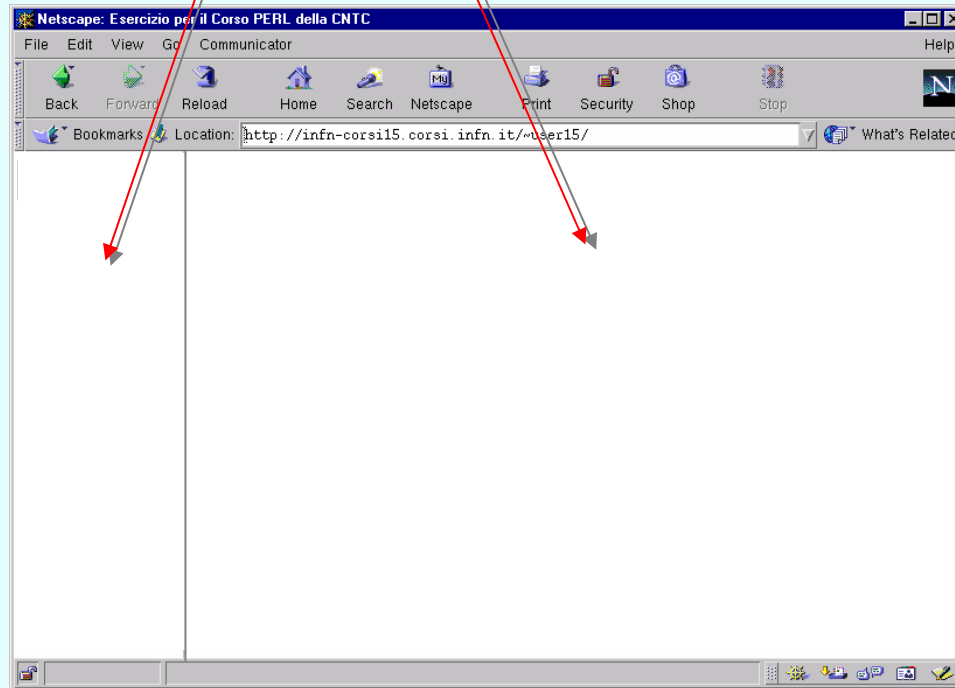


Modifichiamo ora il file **index.html** affinché descriva una pagina WEB dotata di due *frames* distinti.



```
index.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/index.html 317 bytes
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<TITLE>Esercizio per il Corso PERL della CNTC</TITLE>
<FRAMESET COLS="150,*" FRAMEBORDER=NO BORDER=1>

</FRAMESET>
</Html>
```

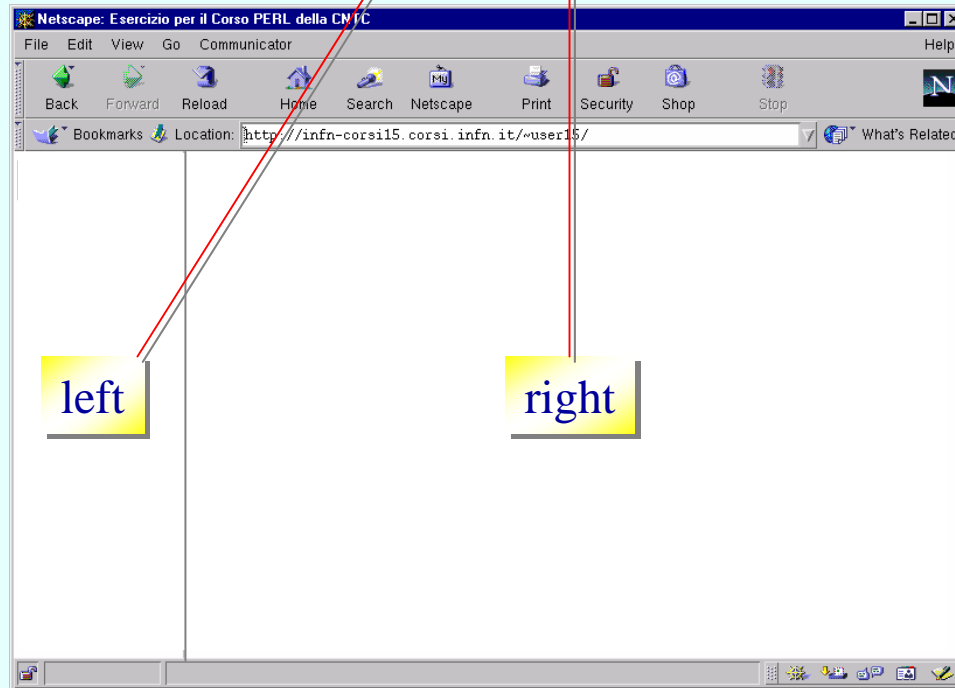


Il frame a sinistra lo facciamo di larghezza fissa, 150 pixel

Il frame a destra lo facciamo di larghezza variabile (sceglie l'utente mediante un *resize* della finestra del browser)



```
index.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/index.html 317 bytes
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<TITLE>Esercizio per il Corso PERL della CNTC</TITLE>
<FRAMESET COLS="150,*" FRAMEBORDER=NO BORDER=1>
  <FRAMESET ROWS="*">
    <FRAME SRC="Menu.html" NAME="left" MARGINWIDTH=3>
    <FRAME SRC="Main.html" NAME="right">
  </FRAMESET>
</FRAMESET>
</Html>
```



Sia il frame a sinistra che il frame a destra sono costituiti da una sola colonna, di larghezza variabile (lasciamo sempre libero l'utente di scegliere)



```
index.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/index.html 317 bytes
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<TITLE>Esercizio per il Corso PERL della CNTC</TITLE>

<FRAMESET COLS="150,*" FRAMEBORDER=NO BORDER=1>
  <FRAMESET ROWS="*">
    <FRAME SRC="Menu.html" NAME="left" MARGINWIDTH=3>
    <FRAME SRC="Main.html" NAME="right">
  </FRAMESET>
</FRAMESET>

</Html>
```

```
Menu.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/Menu.html 194 bytes
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<BODY BGCOLOR="WHITE">
</HEAD>

<TITLE>
WEB interface to a PAW histograms presenter
</TITLE>

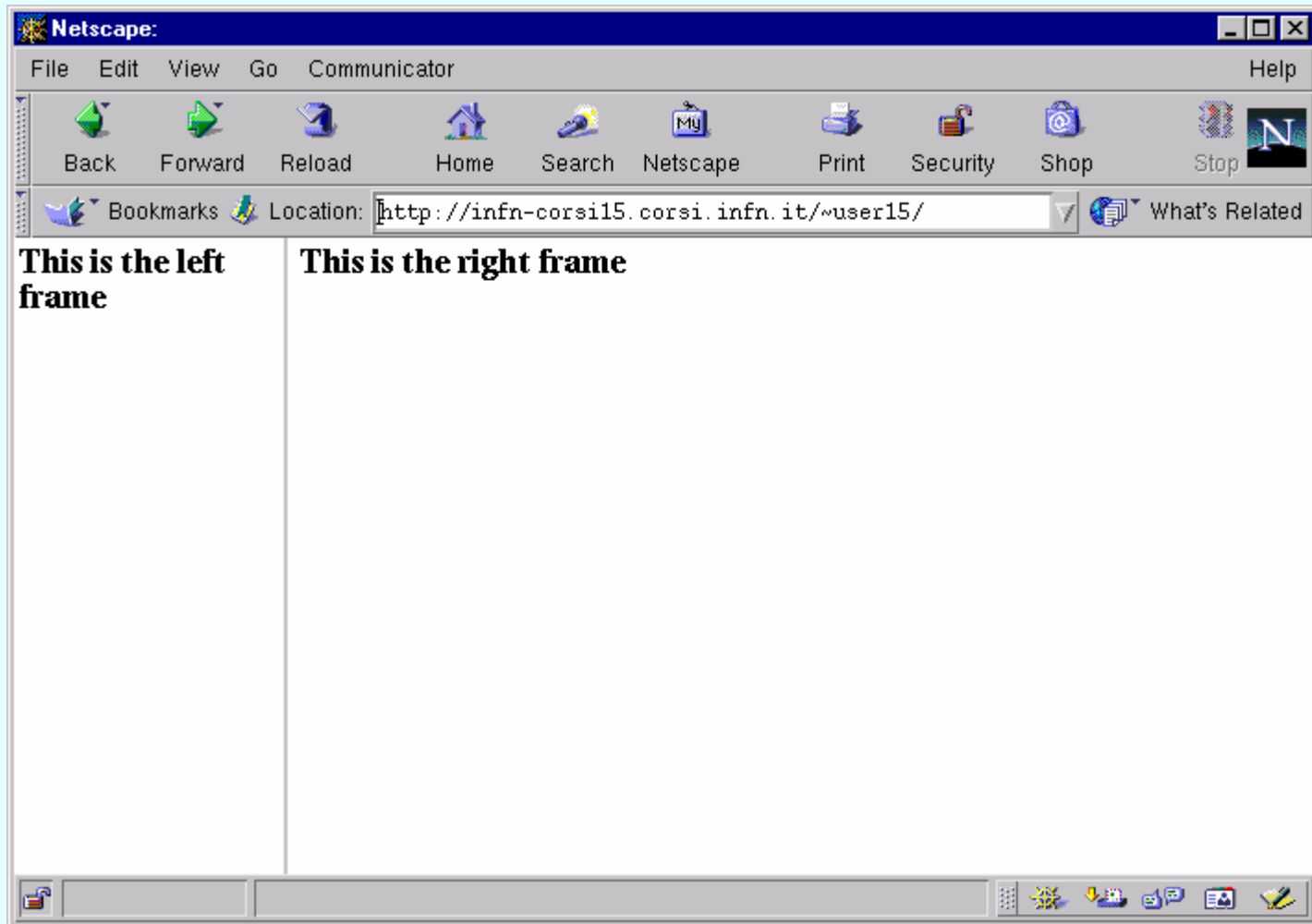
<H1>
This is the left frame
</H1>
```

```
Main.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/Main.html 195 bytes
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<BODY BGCOLOR="WHITE">
</HEAD>

<TITLE>
WEB interface to a PAW histograms presenter
</TITLE>

<H1>
This is the right frame
</H1>
```





Dotiamo ora il nostro frame sinistro di un bottone, associato ad uno script, in modo che quando un utente pigia il bottone in questione, venga attivato lo script.

Definiamo allo scopo un *form* dotato di un solo elemento attivo, il bottone in questione.

```
Menu.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/Menu.html 372 bytes
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<BODY BGCOLOR="WHITE">
</HEAD>

<TITLE>
WEB interface to a PAW histograms presenter
</TITLE>

<Form Action="http://inf-n-corsi15.corsi.infn.it/cgi-bin/user15/HServer.pl"
Method="Get"
Target="right">

</Form>
```

URL dello script da eseguire

Metodo col quale passare degli eventuali argomenti dal form allo script (dettagli fra poco)

Frame al quale lo script deve mandare l'output prodotto (che consisterà, nel nostro caso, di una pagina HTML creata al volo dallo script durante l'esecuzione).



Definito il frame, *contenitore*, dotiamolo di attributi per l'interazione fra l'utente e lo script: usiamo allo scopo l'attributo di un form di tipo *button*.

```
Menu.html
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/Menu.html 372 bytes
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<BODY BGCOLOR="WHITE">
</HEAD>
<TITLE>
WEB interface to a PAW histograms presenter
</TITLE>
<Form Action="http://inf-n-corsi15.corsi.infn.it/cgi-bin/user15/HServer.pl"
Method="Get"
Target="right">
<Input Type="Submit"
Name="ListFiles"
Value="List of existing files">
</Form>
```

Ad un elemento attivo è associato un tipo, un nome che lo identifica ed un eventuale valore.

Archive: hep-th

- new abstracts View
- new abstracts /author Listings
- recent Show Abstract
- help
- lastupdate
- info

Author(s): [] Find

--> cond-mat subject classes

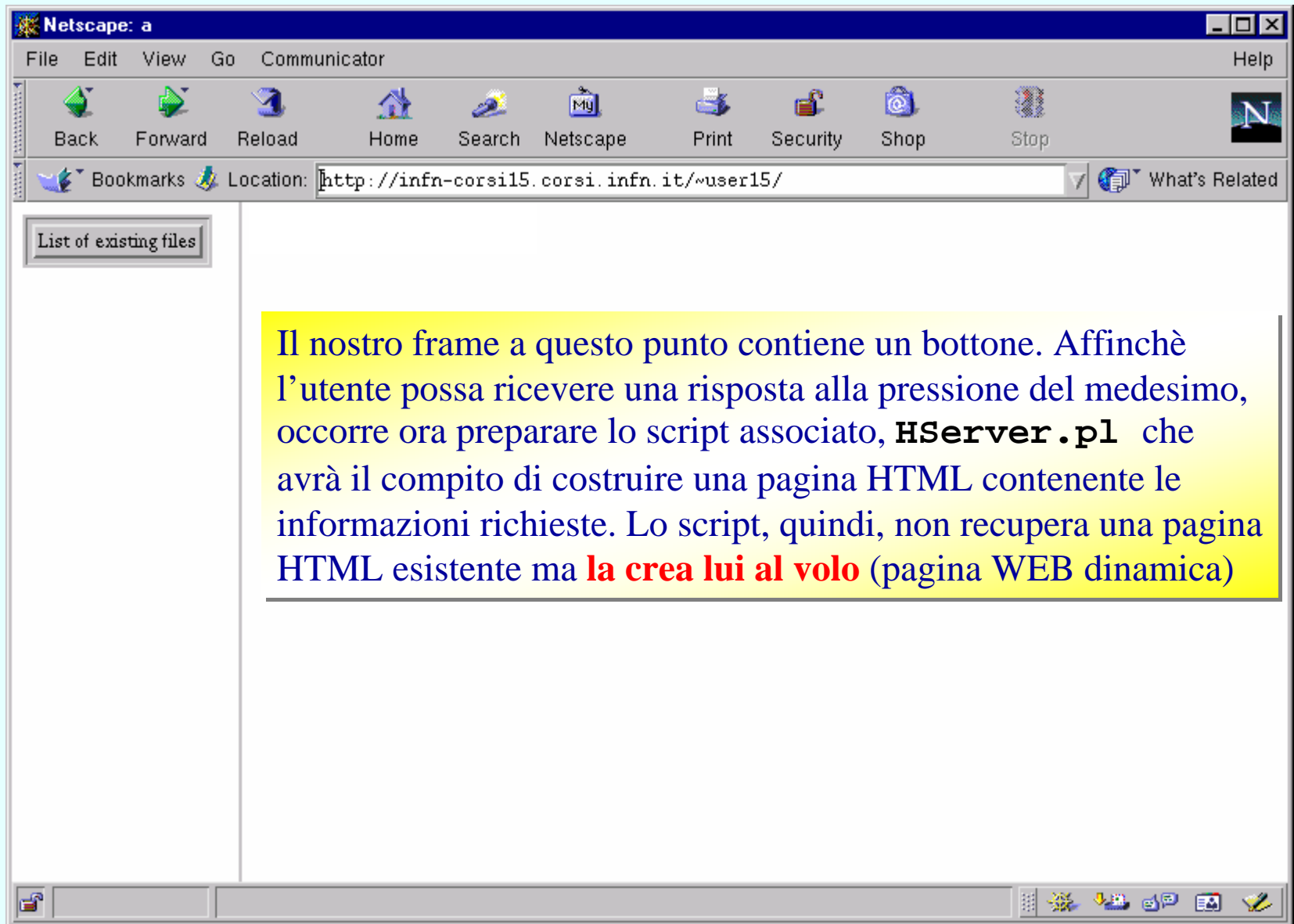
--> physics subject classes

Reset selections to default values.

Type="Selection"

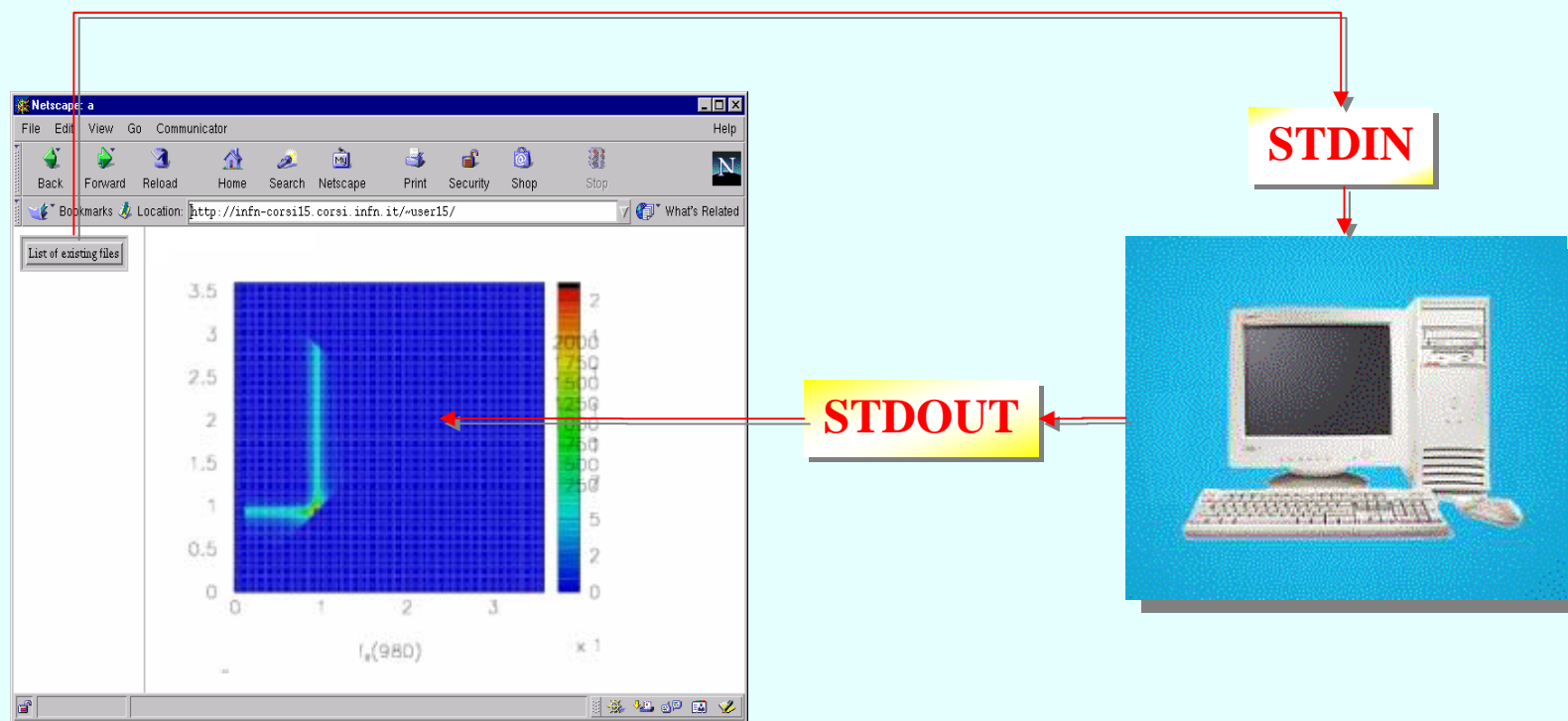
Type="Text"





A scopo introduttivo, mostriamo un esempio di script che abbia come unico effetto quello di scrivere sul frame designato la stringa **"Welcome to HServer.pl!!"**

A questo punto entra in scena per la prima volta il protocollo **CGI**: uno script attivato dal server **WEB** vede come **STDIN** il canale di comunicazione dal browser (riceve input dal canale **TCP/IP**) e l'output che produce (il suo **STDOUT**) viene rimandato allo stesso canale.



Come fa il server a sapere che richiediamo l'esecuzione di un programma e non il listato del medesimo? Cosa dobbiamo fare per informare correttamente il server?



Useremo la definizione formale di **URL**, estesa al caso di esecuzione di uno script mediante il protocollo **CGI**. La definizione di **URL**, nel contesto del formato **HTML** è la seguente:

```
<A HREF="http://infncorsi15.corsi.infn.it/~user15/testo.html">  
  Testo  
</A>
```

Hyperlink tags

Testo associato allo hyperlink

Protocollo di comunicazione richiesto: possibili valori sono file://, http://, ftp://

Indirizzo IP del nodo al quale risponde un WEB server (nella forma risolvibile da un DNS o nella forma ddd.ddd.ddd.ddd)

Testo

Full path name del file contenente la pagina HTML che il server dovrà fornire allorché un utente selezionerà lo hyperlink **Testo**

Questa definizione è tutto ciò che occorre per istruire il browser a fornire ad un WEB server la richiesta di un documento (sia che si tratti di una pagina HTML, di un file PDF o POSTSCRIPT, di un video, un audio o altro. Vediamo il caso in cui il server debba invece invocare uno script al quale servono uno o più parametri in input.



Anzitutto: come fa il server a sapere che gli viene richiesto di *eseguire* uno script e non di *fornire* al browser richiedente il sorgente dello script (che sarebbe il comportamento default)?

Il server ne viene informato grazie al fatto che lo URL per uno script differisce dallo URL per la richiesta di un documento (file HTML, video, audio o altro) per la presenza della stringa (convenzionale) **cgi-bin**.

```
<A HREF="http://inf-n-corsi15.corsi.infn.it/testo.html">  
  Testo  
</A>
```

URL convenzionale per la richiesta di un documento HTML

```
<A HREF="http://inf-n-corsi15.corsi.infn.it/cgi-bin/user15/script.pl">  
  Testo  
</A>
```

URL per la richiesta di esecuzione di uno script, nella fattispecie un programma in PERL, **script.pl**

Ma chi dice al server dove sta, fisicamente, il sorgente dello script **script.pl**?

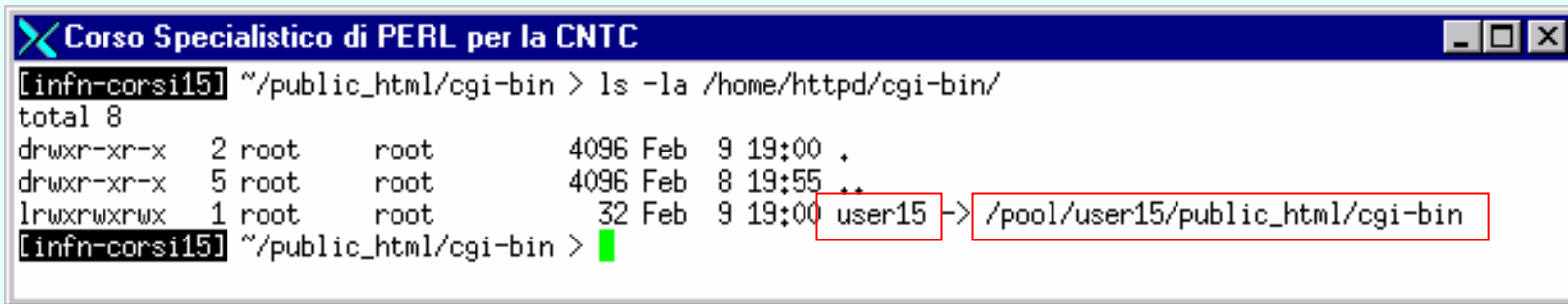
Questa informazione è specificata, di nuovo, nei file di configurazione di Apache



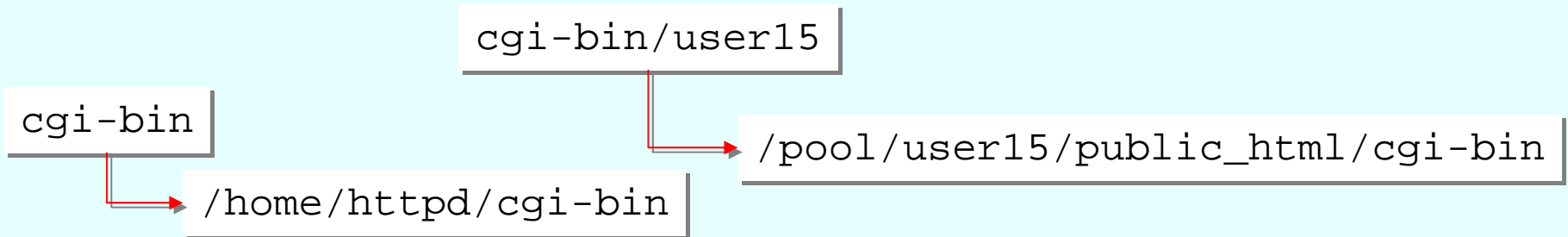
File di configurazione del server httpd (apache)

/etc/httpd/conf/httpd.conf

```
...  
ScriptAlias /cgi-bin/ /home/httpd/cgi-bin  
...  
<Directory /home/httpd/cgi-bin >  
...  
Options ExecCGI FollowSymLinks  
...  
</Directory>
```



```
Corso Specialistico di PERL per la CNTC  
[infn-corsi15] ~/public_html/cgi-bin > ls -la /home/httpd/cgi-bin/  
total 8  
drwxr-xr-x  2 root  root    4096 Feb  9 19:00 .  
drwxr-xr-x  5 root  root    4096 Feb  8 19:55 ..  
lrwxrwxrwx  1 root  root     32 Feb  9 19:00 user15 -> /pool/user15/public_html/cgi-bin  
[infn-corsi15] ~/public_html/cgi-bin >
```



Alla fine di questo complesso gioco di scatole cinesi (ideato per massimizzare il controllo sulla security delle richieste al server) uno **URL** per attivare uno script avrà una forma del tipo:

```
http://infncorsi15.corsi.infn.it/cgi-bin/user15/script.pl
```

```
cgi-bin/user15/script.pl
```

```
/pool/user15/public_html/cgi-bin/script.pl
```

È di fondamentale importanza a questo punto avere ben chiaro chi esegue lo script: lo esegue il **server** tramite l'azione di un demone sempre attivo ed in ascolto sulla porta 80 (o quella definita nel file di configurazione di apache). Il demone, *httpd*, agisce in qualità di **user nobody**, un **user** sprovvisto di ogni privilegio se non quelli che gli vengono garantiti dal file di configurazione di apache stesso. Ci si garantisce in questo modo un buon margine di sicurezza nei confronti di potenziali hackers. La conseguenza è che molti files, visibili in modo interattivo da un utente generico possono non essere visibili o accessibili allo **user nobody**.



Ecco il nostro primo script **PERL** che agisce attivato dal demone **httpd**, sempre attivo in un **WEB** server. Nel caso specifico delle macchine del laboratorio, lo script deve stare nella directory **public_html/cgi-bin**, perché il server è configurato in modo tale da eseguire programmi solo da questa particolare area. (Vedi il file **httpd.conf**).

```
HServer.pl
File Edit Search Preferences Shell Macro Windows
/pool/user15/public_html/cgi-bin/HServer.pl 350 bytes
#!/usr/bin/perl
#
# Author: D. Menasce
# WEB interface for a PAW Histogram server
# Corso specialistico di PERL per la CNTC (Marzo 2001)
#
#=====
#
print <<End_Of_Text ;
Content-type: text/html

<HTML>
<Body BgColor="White">

<H1>
Welcome to HServer.pl!!
</H1>

End_Of_Text
```

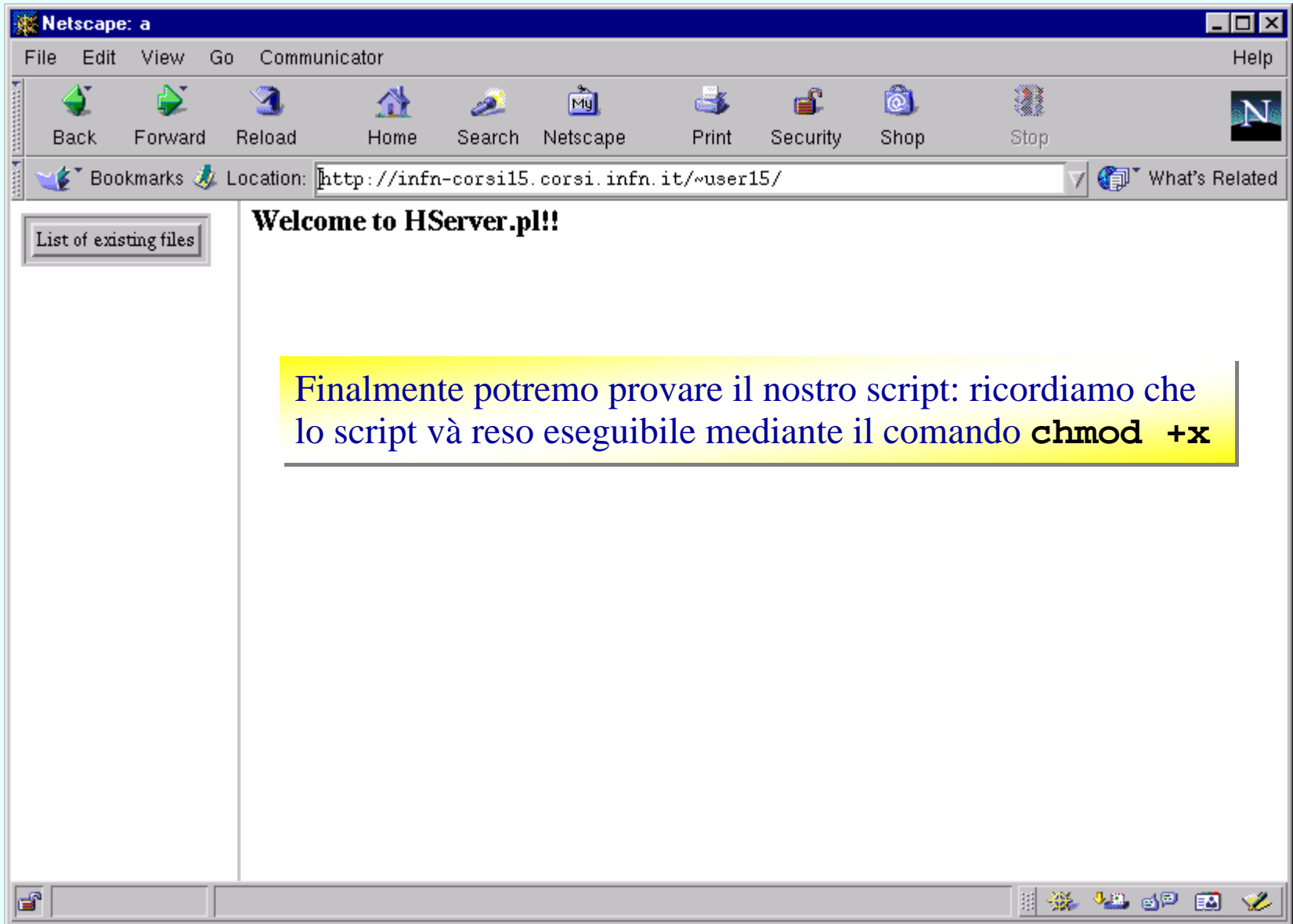
Il protocollo **CGI** prevede, inoltre, che un documento ricevuto da un browser da parte di un server che lo ha fornito, contenga, come prima riga, la specifica di tipo di documento in arrivo. Nel nostro caso si tratta di una pagina **WEB**, per cui faremo in modo che la prima riga che lo script manda allo **STDOUT** sia:

Content-type: text/html\n\n

Subito dopo specificheremo il tag **<HTML>** e tutto il resto del contenuto della pagina **WEB**

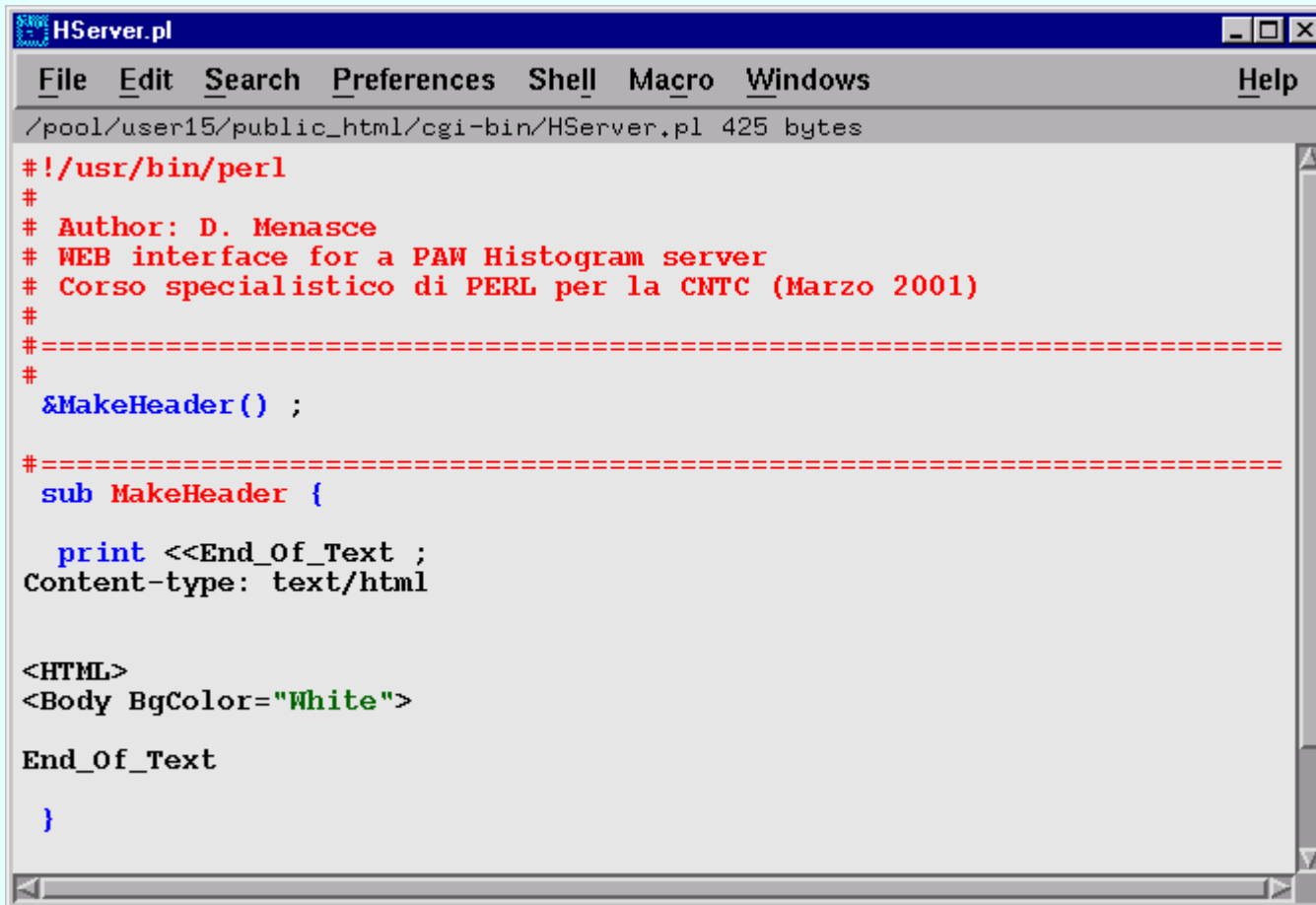
Il protocollo **CGI** prevede che il **mime-type** (la specifica di che tipo di documento si tratta) sia seguita da **due righe vuote**, senza neppure un carattere blank!!! **Attenzione.**





Capito il principio di base, cominciamo a modificare lo script per dotarlo della funzionalità che ci interessa implementare: uno histogram browser nell'ambito di **PAW**.

Definiamo allo scopo una prima funzione, **MakeHeader**, che crei l'intestazione del documento



```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 425 bytes
#!/usr/bin/perl
#
# Author: D. Menasce
# WEB interface for a PAW Histogram server
# Corso specialistico di PERL per la CNTC (Marzo 2001)
#
#=====
#
&MakeHeader() ;

#=====
sub MakeHeader {
    print <<End_Of_Text ;
Content-type: text/html

<HTML>
<Body BgColor="White">

End_Of_Text
}
```



```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 906 bytes
#!/usr/bin/perl
#
# Author: D. Menasce
# WEB interface for a PAW Histogram server
# Corso specialistico di PERL per la CNTC (Marzo 2001)
#
#=====
#
&MakeHeader() ;
&Initialize() ;
#=====

sub MakeHeader {

    print <<End_Of_Text ;
    Content-type: text/html

<HTML>
<Body BgColor="White">

End_Of_Text
}

#=====

sub Initialize {

    $HisDir = "/pool/user15/public_html/data/*.his" ;

}

#=====
```

Dopodichè invocheremo una funzione di inizializzazione, che avrà lo scopo di definire variabili: tutte le caratteristiche di funzionamento dello script le metteremo qui per averne un controllo semplificato.

Per ora ci limitiamo a definire una sola variabile contenente il full path-name della directory che contiene i file istogrammi che lo script dovrà servire.



```
HServer.pl
File Edit Search Preferences Shell
/pool/user15/public_html/cgi-bin/HServer.pl

#!/usr/bin/perl
#
# Author: D. Menasce
# WEB interface for a PAW Histogram server
# Corso specialistico di PERL per la CNTC (Marzo 2004)
#
#=====
#
&MakeHeader() ;
&Initialize() ;
&ShowFileList();
#=====

sub MakeHeader {

    print <<End_Of_Text ;
Content-type: text/html

<HTML>
<Body BgColor="White">

End_Of_Text

}

#=====

sub Initialize {

    $HisDir = "/pool/user15/public_html/data/*.his" ;

}

#=====
```

Invochiamo ora una funzione che listi i files presenti nella directory specificata dalla variabile `$HisDir`.

```
sub ShowFileList {

    print <<EOT ;
<H3>
<Center>
    List of histograms in the $HisDir directory
</Center>

<ol>
EOT

    foreach $File (glob "$HisDir") {
        print(" <LI> $File\n" );
    }

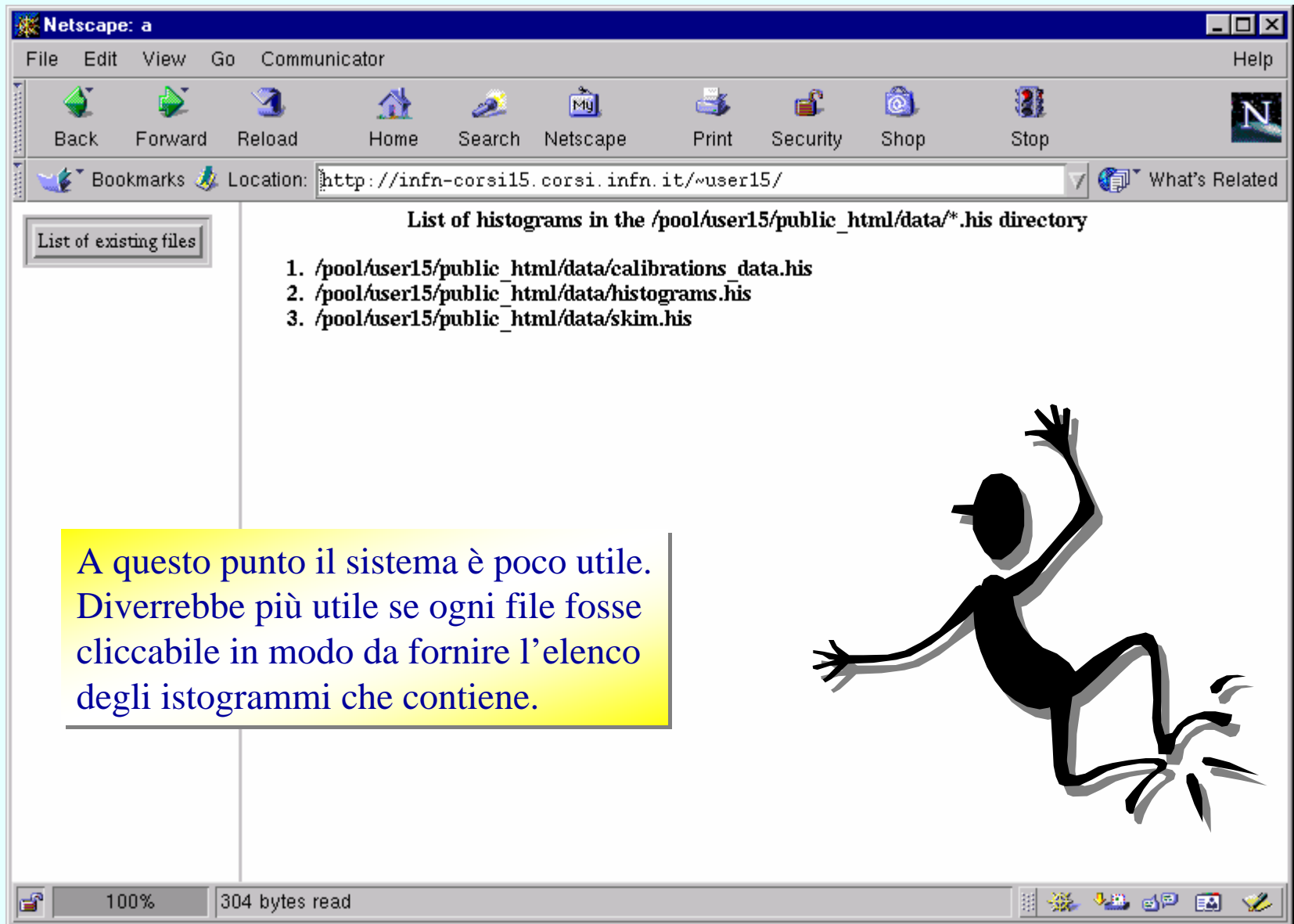
    print(" </OL>\n" );

}

}
```

Un ciclo iterativo mediante il comando *glob* ci fornisce la lista richiesta. Il comando *print* spedisce la lista allo **STDOUT** e quindi, in accordo al protocollo **CGI**, al frame del browser specificato dalla chiamata. ⓘ





Supponiamo, come nel caso del nostro esempio, che lo script attivato dal server in seguito alla richiesta di un browser, debba fornire l'elenco dei file in una directory. Occorre quindi che lo script possa acquisire il *path name* della directory in input (nome che verrà fornito con un qualche meccanismo che presto esplicheremo, dal browser stesso). In questo caso il protocollo CGI prevede che lo **URL** assuma la seguente forma:

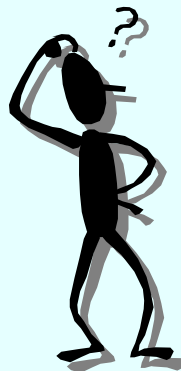
```
<A HREF=" URL?Elenco-dei-parametri "> Testo </A>
```

```
nome-parametro=valore&altro-parametro=valore&...
```

Coppie *chiave-valore* separate da &

```
http://infncorsi15.corsi.infn.it/cgi-bin/user15
```

Bene: ma come fa lo script ad acquisire questo elenco dei parametri?



Quando il WEB server (uno dei demoni di apache, httpd) riceve una richiesta di servizio sulla porta dalla quale ascolta (generalmente la porta 80), fa partire un sottoprocesso nel quale viene eseguito lo script eventualmente richiesto. Nell'ambito di questo sottoprocesso vengono istanziate tutta una serie di variabili ambientali, una delle quali contiene la lista dei parametri forniti in ingresso mediante il meccanismo visto prima.

```
xterm
[infncorsi15] ~ > ps aux | grep http
root      583  0.0  0.9 2580 1232 ?        S    Feb08   0:00 httpd
nobody    9446  0.0  0.9 2816 1272 ?        S    Feb11   0:00 httpd
nobody    9447  0.0  0.9 2816 1272 ?        S    Feb11   0:00 httpd
nobody    9448  0.0  0.9 2816 1272 ?        S    Feb11   0:00 httpd
nobody    9449  0.0  0.9 2816 1272 ?        S    Feb11   0:00 httpd
nobody    9450  0.0  0.9 2816 1272 ?        S    Feb11   0:00 httpd
nobody    9451  0.0  0.9 2816 1268 ?        S    Feb11   0:00 httpd
nobody    9452  0.0  0.9 2820 1280 ?        S    Feb11   0:00 httpd
nobody    9453  0.0  0.9 2816 1280 ?        S    Feb11   0:00 httpd
user15    21675 0.0  0.3 1168  456 pts/0    S    12:47   0:00 grep http
[infncorsi15] ~ >
```

```
test.pl
File Edit Search Preferences Shell Macro Windows Help
'user15/public_html/cgi-bin/test.pl 361 bytes
#!/usr/bin/perl

print <<EOT ;
Content-type: text/html

<H2>
Partial list of important environment variables: <P>

<PRE>

REQUEST_METHOD = $ENV{REQUEST_METHOD}
SERVER_PORT    = $ENV{SERVER_PORT}
DOCUMENT_ROOT  = $ENV{DOCUMENT_ROOT}
QUERY_STRING   = $ENV{QUERY_STRING}
REMOTE_HOST    = $ENV{REMOTE_HOST}
REMOTE_ADDR    = $ENV{REMOTE_ADDR}

</PRE>
</HTML>

EOT
```

Elenco (**parziale**) delle variabili definite nell'ambito di un processo istanziato dal demone httpd



```
xterm
[infncorsi15] ~/public_html/cgi-bin > ./test.pl
Content-type: text/html

<H2>
Partial list of important environment variables: <P>

<PRE>

REQUEST_METHOD =
SERVER_PORT    =
DOCUMENT_ROOT  =
QUERY_STRING   =
REMOTE_HOST    =
REMOTE_ADDR    =

</PRE>
</HTML>

[infncorsi15] ~/public_html/cgi-bin > █
```

Se eseguiamo lo script in una shell interattiva, *non* vi saranno definite le variabili ambientali.

Se invece invochiamo lo script tramite il WEB server ecco che le variabili vengono istanziate ai valori che loro competono.

In particolare QUERY_STRING contiene il parametro associato all'invocazione dello script




```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl line 74, col 0, 1234 bytes
#=====

sub ShowFileList {

    print <<EOT ;
<H3>
<Center>
    List of histograms in the $HisDir directory
</Center>

<OL>
EOT

    foreach $File (glob "$HisDir") {
        $Link = "http://${ServerMachine}/${ServerScript}" . "?" .
                "Action=ShowHistograms" . "&" .
                "File=$File" ;
        print(" <LI> <A HREF=$Link> $File </A>\n" ) ;
    }
    print(" </OL>\n" ) ;
}

sub Initialize {

    $User          = "15" ;
    $HisDir        = "/pool/user$User/public_html/data/*.his" ;
    $ServerMachine = "inf-n-corsi${User}.corsi.inf-n.it" ;
    $ServerScript  = "cgi-bin/user${User}/HServer.pl" ;

}
```

Ecco ora la modifica da apportare allo script in modo che produca in output un testo corrispondente ad uno hyperlink

Notiamo il fatto che lo script che invochiamo per eseguire la nuova funzione è sempre **HServer.pl**, lo *script corrente*. Vedremo poi come modificarlo affinché possa agire nelle due modalità richieste.

```
sub Initialize {

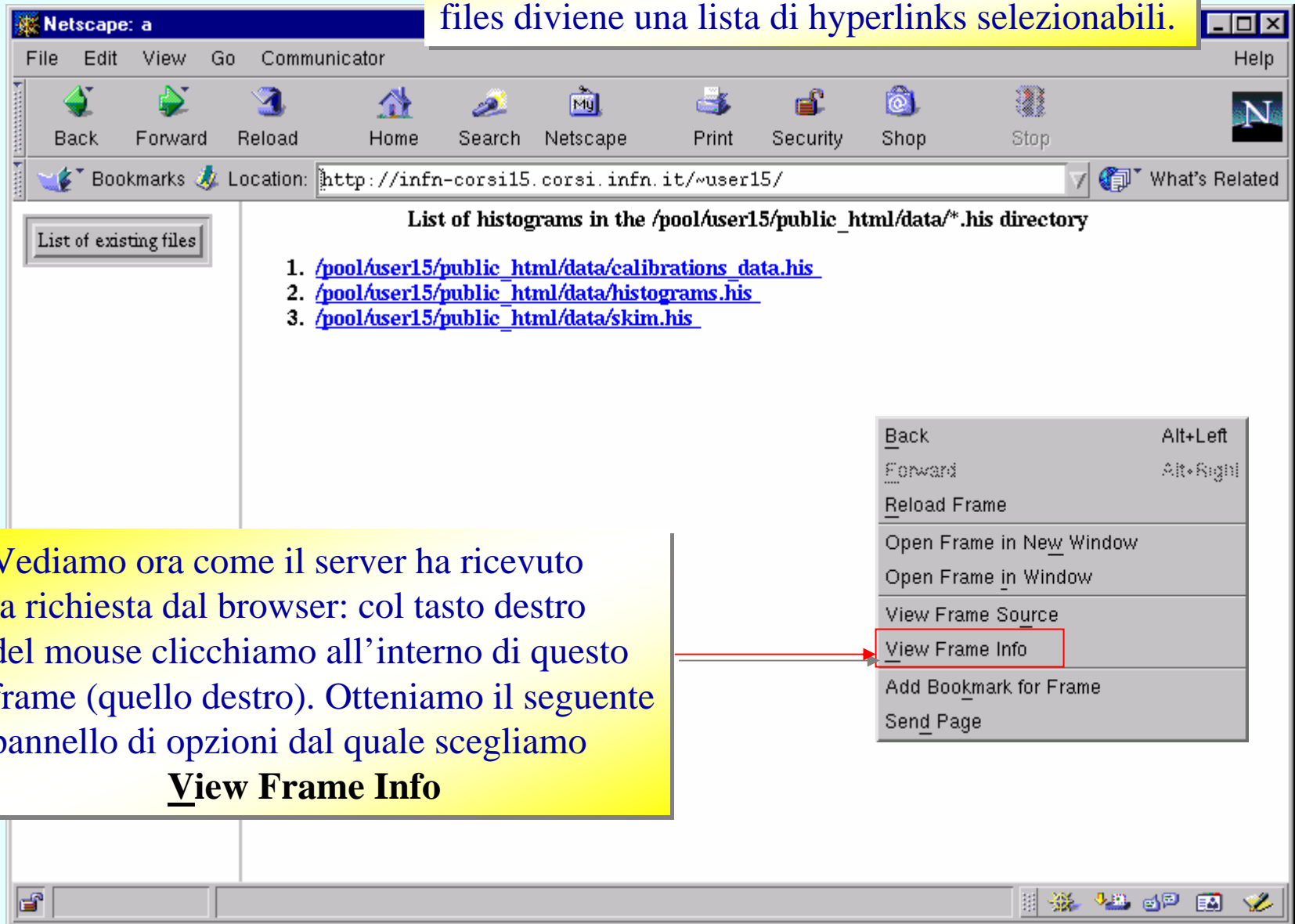
    $User          = "15" ;
    $HisDir        = "/pool/user$User/public_html/data/*.his" ;
    $ServerMachine = "inf-n-corsi${User}.corsi.inf-n.it" ;
    $ServerScript  = "cgi-bin/user${User}/HServer.pl" ;

}
```

La **QUERY_STRING** sarà, in questo esempio, **Action=ShowHistograms&File=/pool/user15/...**



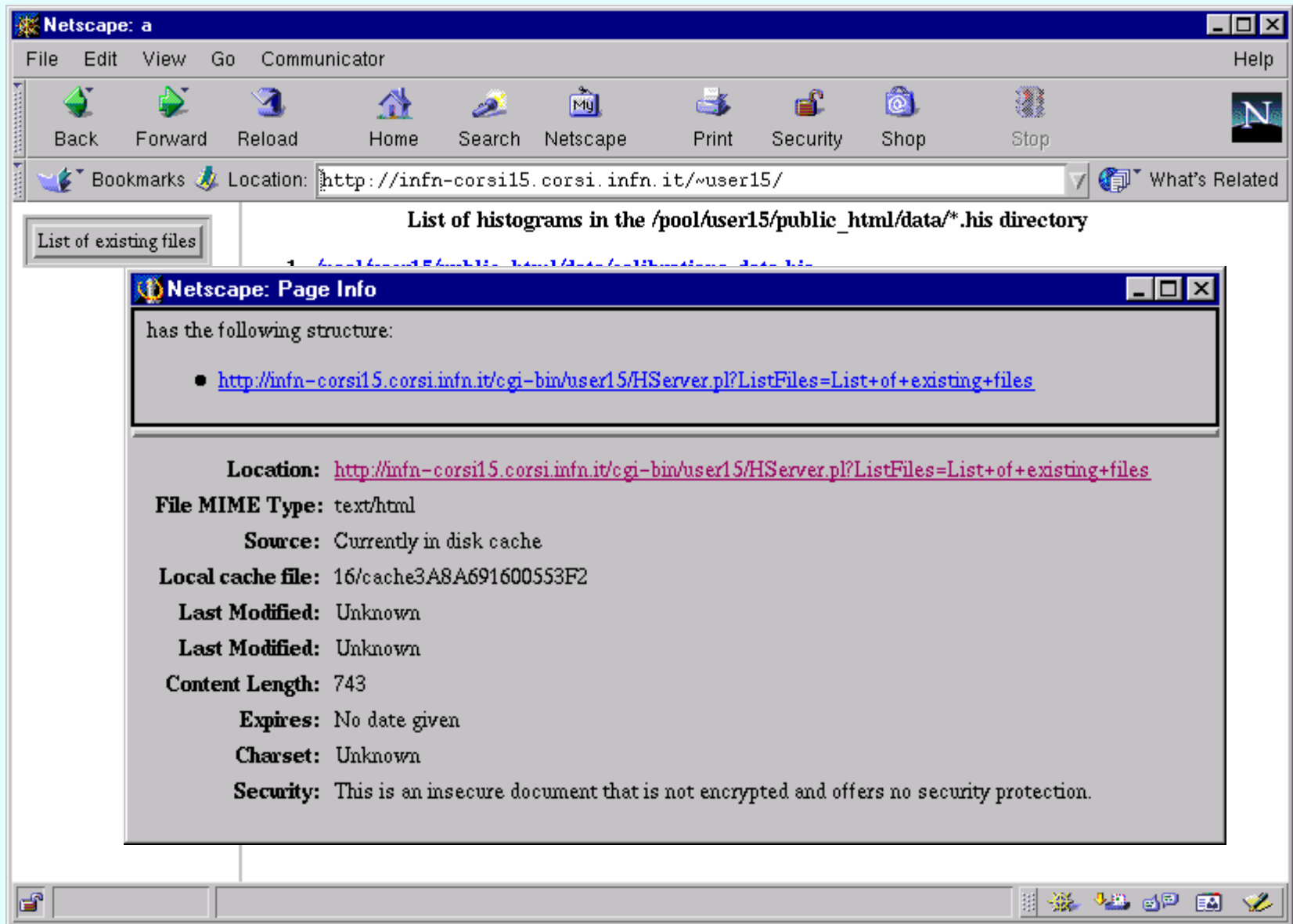
Se eseguiamo lo script ora, ecco che l'elenco dei files diviene una lista di hyperlinks selezionabili.

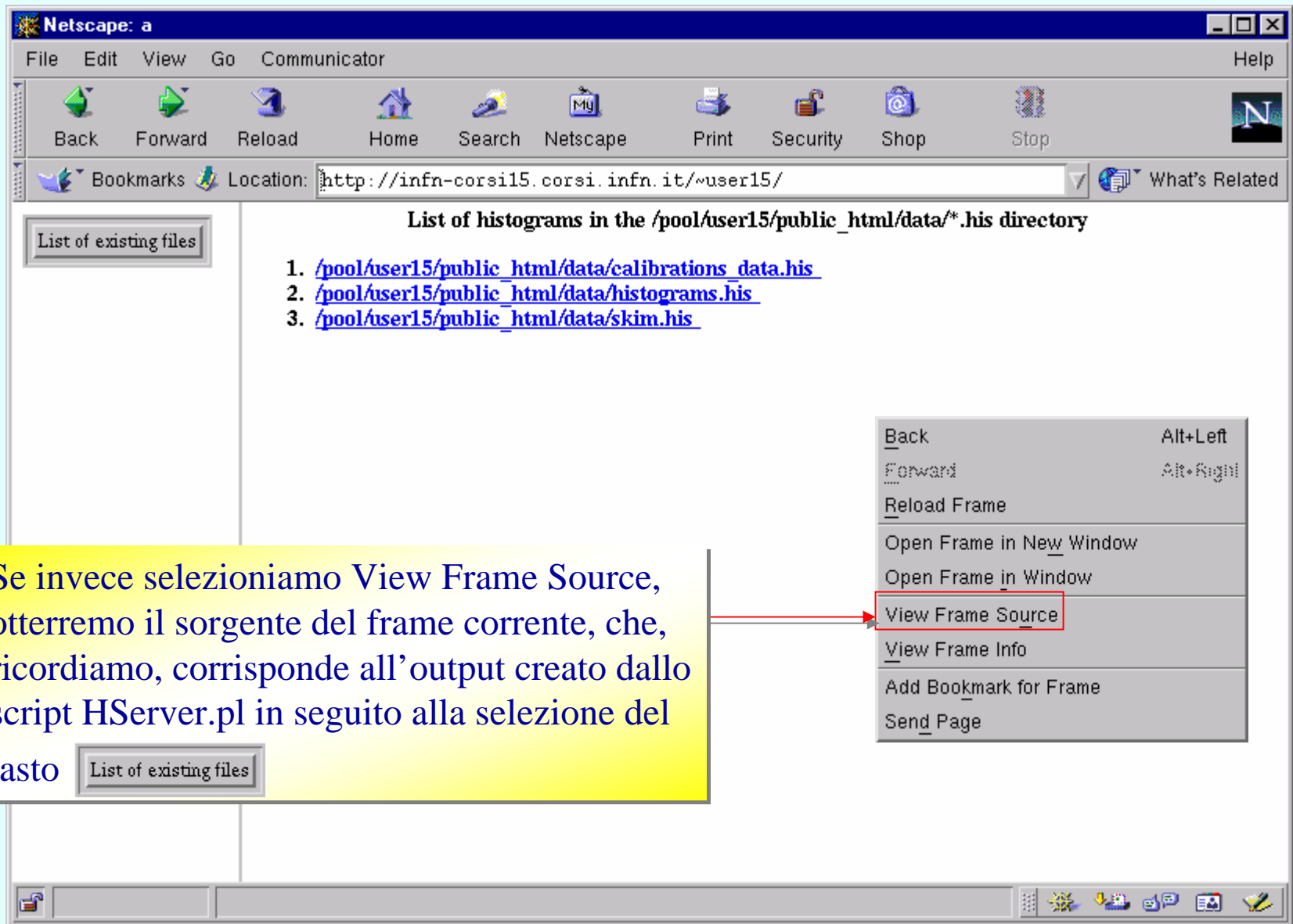


Vediamo ora come il server ha ricevuto la richiesta dal browser: col tasto destro del mouse clicchiamo all'interno di questo frame (quello destro). Otteniamo il seguente pannello di opzioni dal quale scegliamo

View Frame Info







Se invece selezioniamo View Frame Source, otterremo il sorgente del frame corrente, che, ricordiamo, corrisponde all'output creato dallo script HServer.pl in seguito alla selezione del tasto



```
Netscape: Source of: http://infn-corsi15.corsi.infn.it/cgi-bin/user15/HServer.pl?ListFile...

<HTML>
<Body BgColor="white">

<H3>
  <Center>
    List of histograms in the /pool/user15/public_html/data/*.his directory
  </Center>

<OL>
  <LI> <A HREF=http://infn-corsi15.corsi.infn.it/cgi-bin/user15/Hserver.pl?Action=ShowHistograms
  <LI> <A HREF=http://infn-corsi15.corsi.infn.it/cgi-bin/user15/Hserver.pl?Action=ShowHistograms
  <LI> <A HREF=http://infn-corsi15.corsi.infn.it/cgi-bin/user15/Hserver.pl?Action=ShowHistograms
</OL>
```

Abbiamo visto che lo script riceve il parametro associato tramite la variabile ambientale QUERY_STRING: occorre che lo script sia dotato di una opportuna funzione che sappia interpretare l'elenco dei parametri in ingresso

```
tograms&File=/pool/user15/public_html/data/calibrations_data.his> /pool/user15/public_html/data/ca
tograms&File=/pool/user15/public_html/data/histograms.his> /pool/user15/public_html/data/histogram
tograms&File=/pool/user15/public_html/data/skim.his> /pool/user15/public_html/data/skim.his </A>
```



```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 2090 bytes
#!/usr/bin/perl
#
# Author: D. Menasce
# WEB interface for a PAW Histogram server
# Corso specialistico di PERL per la CNTC (Marzo 2001)
#
#=====
#
&MakeHeader() ;
&Initialize() ;
&DecodeQueryString() ;
&ShowFileList() ;
```

```
sub DecodeQueryString {
    my $request_method = $ENV{'REQUEST_METHOD'} ;
    if ($request_method eq "GET") {
        $query_string = $ENV{'QUERY_STRING'} ;
    } else {
        my $size = $ENV{'CONTENT_LENGTH'} ;
        read (STDIN, $query_string, $size) ;
    }
    my @pairs = split (/&/, $query_string) ;
    foreach my $key_val ( @pairs ) {
        ($key, $value) = split ( /=/, $key_val ) ;
        $value =~ tr/+/ / ;
        $value =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack ("C", hex ($1))/eg ;
        $$key = $value ;
    }
}
```

Scriviamo quindi una funzione che spezzi la **QUERY_STRING** nell'insieme di coppie chiave valore e per ognuna di esse definisca una variabile il cui nome sia la chiave e di contenuto abbia il valore.

Di conseguenza verrà definita una variabile **\$Action = "ShowHistograms"**

Ricordiamo che la **QUERY_STRING** che lo script riceve è della forma **Action=ShowHistograms&File=/pool/user15/...**



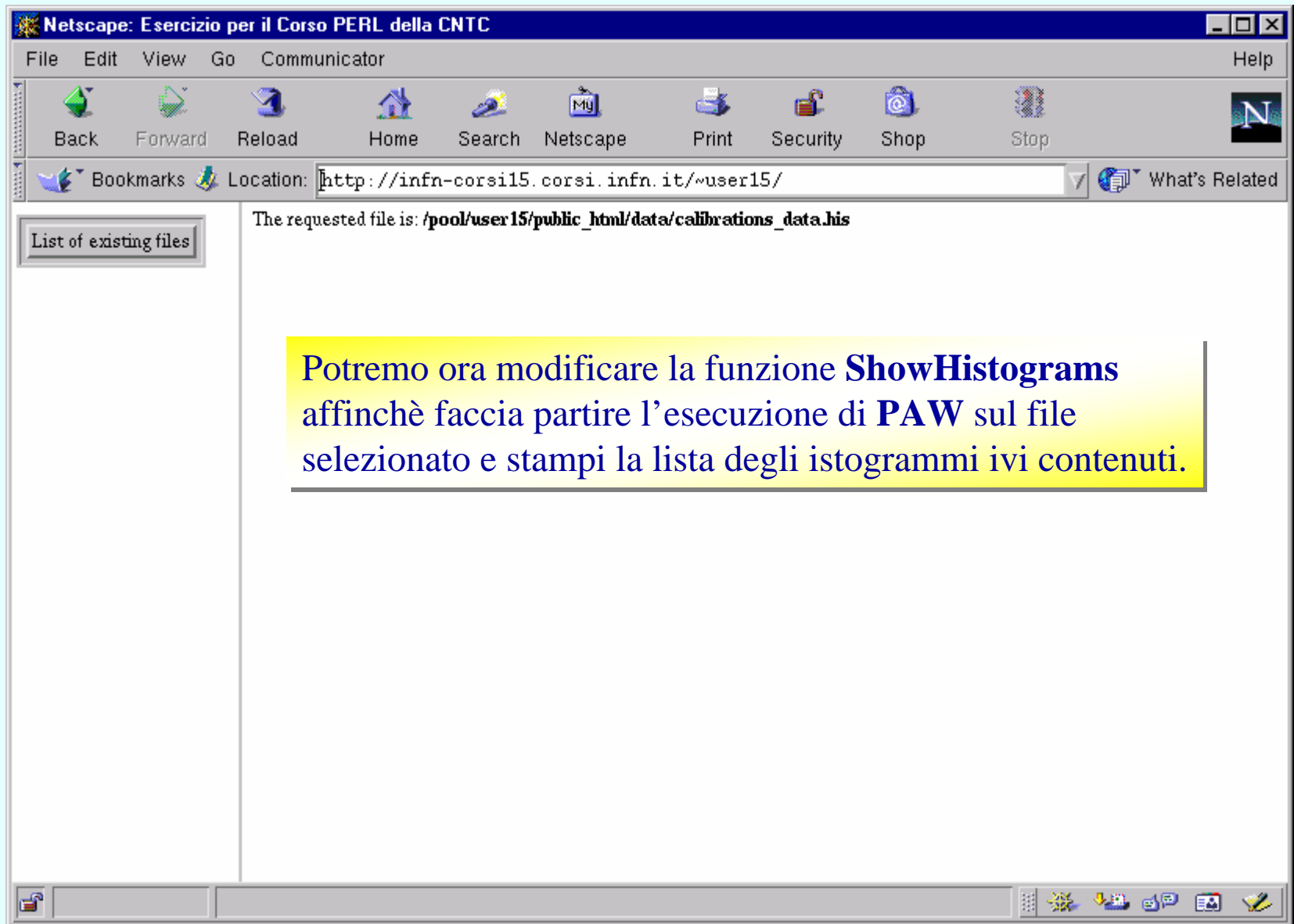
```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 2090 bytes
#!/usr/bin/perl
#
# Author: D. Menasce
# WEB interface for a PAW Histogram server
# Corso specialistico di PERL per la CNTC (Marzo 2001)
#
#=====
#
&MakeHeader() ;
&Initialize() ;
&DecodeQueryString() ;

if ($Action =~ m/ShowHistograms/) {
    &ShowHistograms() ;
} else {
    &ShowFileList() ;
}

sub ShowHistograms {
    print("The requested file is: <B>$File</B>\n") ;
}
```

Potremo a questo punto usare il valore di questa variabile, **\$Action** per ridirigere il flusso del programma in funzione della richiesta proveniente dal browser. Prima lo script eseguiva solamente la funzione **ShowFileList**, ora può alternativamente eseguire la funzione **ShowHistograms** (che per ora scrive sul terminale solamente una frase a scopo dimostrativo della funzionalità dello script).






```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 2543 bytes
#=====
sub ShowHistograms {
    &CreateKumacFile() ;
}
#=====
```

Come abbiamo visto nell'esempio  , ci occorre una funzione che crei un file **KUIP** con le istruzioni a **PAW** di listare gli istogrammi di un file.



```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 2543 bytes

#=====

sub ShowHistograms {
    &CreateKumacFile() ;
}

#=====

sub Initialize {
    $User          = "15" ;
    $HisDir         = "/pool/user$User/public_html/data/*.his" ;
    $ServerMachine = "infn-corsi${User}.corsi.infn.it" ;
    $ServerScript  = "cgi-bin/user${User}/HServer.pl" ;
    $KumacFile     = "/tmp/ShowList.kumac" ;
}

#=====

sub CreateKumacFile {
    open( KUM, ">$KumacFile" ) ;
    print KUM<<End_Of_Text ;
Macro ShowList

    Message File: <Font Color="Red">$File</Font>
    Message

    His/File 1 $File
    H/Lis

End_Of_Text

    close(KUM) ;
}

#=====
```

```
sub Initialize {
    $User          = "15" ;
    $HisDir         = "/pool/user$User/public_html/data/*.his" ;
    $ServerMachine = "infn-corsi${User}.corsi.infn.it" ;
    $ServerScript  = "cgi-bin/user${User}/HServer.pl" ;
    $KumacFile     = "/tmp/ShowList.kumac" ;
}
```

Il file temporaneo, contenente lo script **KUIP**, lo facciamo creare nella zona **/tmp**, zona alla quale l'utente **nobody**, utente *owner* del processo demone **httpd** (il **WEB** server), ha accesso in scrittura

Notiamo come possiamo liberamente inserire variabili PERL nello script **KUIP**



```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 2543 bytes
#=====
sub ShowHistograms {
    &CreateKumacFile() ;
    print("<PRE>\n") ;
}
#=====
sub CreateKumacFile {
    open( KUM, ">$KumacFile" ) ;
    print KUM<<End_Of_Text ;
Macro ShowList

    Message File: <Font Color="Red">$File</Font>
    Message

    His/File 1 $File
    H/Lis

End_Of_Text

    close(KUM) ;
}

```

Sulla pagina HTML, output dello script, aggiungiamo il tag **<PRE>** che predispone la stampa *verbatim* del testo seguente (mediante fonte non proporzionale, che conserva gli incolonnamenti). Come esercizio si provi ad omettere questo tag.



```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 2543 bytes
#=====
sub ShowHistograms {
    &CreateKumacFile() ;
    print("<PRE>\n") ;
    $ENV{HOME} = "/tmp" ;
}
#=====
sub CreateKumacFile {
    open( KUM, ">${KumacFile}" ) ;
    print KUM<<End_Of_Text ;
Macro ShowList

    Message File: <Font Color="Red">${File}</Font>
    Message

    His/File 1 ${File}
    H/Lis

End_Of_Text

    close(KUM) ;
}
}
```

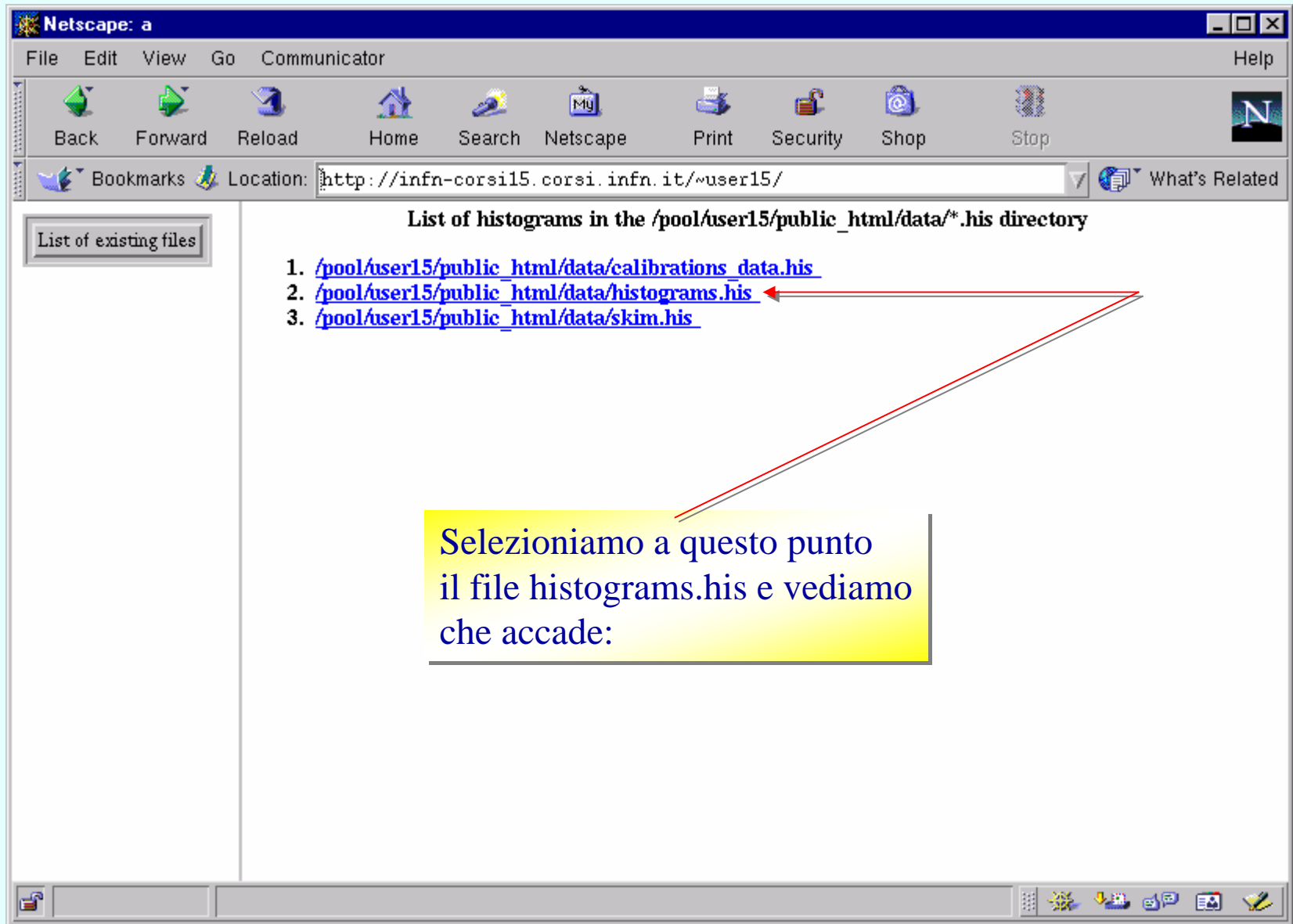
PAW pretende di scrivere nella home area dell'utente che lo lancia il file **last.kumac**. Per determinare la **home** area usa la variabile ambientale **HOME**, che, nel caso dell'utente **nobody** non risulta definita. Provvediamo a ciò definendo la variabile **HOME** in modo che punti alla solita area **/tmp**.



```
HServer.pl
File Edit Search Preferences Shell Macro Windows Help
/pool/user15/public_html/cgi-bin/HServer.pl 2543 bytes
#=====
sub ShowHistograms {
    &CreateKumacFile() ;
    print("<PRE>\n") ;
    $ENV{HOME} = "/tmp" ;
    open( PAW, "/cern/pro/bin/pawX11 -b $KumacFile |" ) ;
    while (<PAW>) {print}
    close(PAW) ;
}
#=====
sub CreateKumacFile {
    open( KUM, ">$KumacFile" ) ;
    print KUM<<End_Of_Text ;
Macro ShowList
    Message File: <Font Color="Red">$File</Font>
    Message
    His/File 1 $File
    H/Lis
End_Of_Text
    close(KUM) ;
}
}
```

Lanciamo l'esecuzione di PAW. Poiché l'utente **nobody** non ha definito nella variabile **PATH** il percorso per l'eseguibile, lo forniamo noi per esteso. Lo lanciamo in modalità batch dandogli il nome dello script **KUIP** da eseguire per fornire l'output voluto.





Netscape: Esercizio per il Corso PERL della CNTC

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: <http://inf-n-corsi15.corsi.infn.it/~user15/> What's Related

List of existing files

```

Version 1.26/04 of HIG2 started
*** No default PAWLOGON file "/tmp/.pawlogon.kumac" found
File: /pool/user15/public_html/data/histograms.his

===> Directory :
  201 (2)  NR
 1201 (2)  Phases NR
  202 (2)  <[r](770)
 1202 (2)  Phases <[r](770)
  203 (2)  f?2!(1275)
 1203 (2)  Phases f?2!(1275)
  204 (2)  f?0!(980)
 1204 (2)  Phases f?0!(980)
  205 (2)  s?0!(1475)
 1205 (2)  Phases s?0!(1475)
  206 (2)  <[r](1450)
 1206 (2)  Phases <[r](1450)
  207 (2)  f?0!(400)
 1207 (2)  Phases f?0!(400)
  208 (2)  f?0!(1300)
 1208 (2)  Phases f?0!(1300)
  401 (2)  1
  402 (2)  2
 8000 (1)  Input file: master_fitter_genric_ds_ppp.his  output file: ppp_effi_ds.his
 1313 (1)  Dalitz Dati
 1314 (1)  Dalitz Dati
 1510 (2)  Dalitz Dati
 1514 (2)  Dalitz Effi  Check
 2001 (1)  <[PPP] mass (Montecarlo)
 1503 (2)  ds side bands
 1501 (2)  ds model
 1000 (1)  Full parameters space description
 3201 (1)  NR
 3202 (1)  <[r](770)
 3203 (1)  f?2!(1275)
 3204 (1)  f?0!(980)
 3205 (1)  s?0!(1475)
 3206 (1)  <[r](1450)
 3207 (1)  f?0!(400)
 3208 (1)  f?0!(1300)
 1515 (2)  Dalitz Adaptive Binning
 1505 (1)  Initial MinimC Parameters
 1506 (1)  Fit  MinimC Parameters
 1507 (1)  Fit Informations
 1504 (2)  Fit Dalitz function (S+B)
 1519 (2)  Fit Dalitz function (S)
 1516 (2)  pure background function from full fit

```

Il giochino a questo punto si ripete: vogliamo rendere ogni istogramma qui indicato con una linea di testo, un opportuno [hyperlink](#) in modo che un semplice click ci permetta di visionarlo in modo grafico nel browser!



```

sub ShowHistograms {
    &CreateKumacFile() ;
    print("<PRE>\n") ;
    $ENV{HOME} = "/tmp" ;

    open( PAW, "/cern/pro/bin/pawX11 -b $KumacFile |" ) ;
    while (<PAW>) {
        if ( m/^\s+(\d+)\s+(\d+)\s+(.+)/ ) {
            $HisId = $1 ;
            $Link = "http://${ServerMachine}/${ServerScript}" . "?" .
                "Action=PrintHistogram" . "&" .
                "HisId=$HisId" . "&" .
                "File=$File" ;
            s|^\(\s+\d+\s+(\d+)\s+(.*)|\$2| ;
        }
        print ;
    }
    close(PAW) ;
}

```

201 (1) Titolo dell'istogramma

Mediante questa *regexp*, determiniamo l'ID di un istogramma (\$1, prima coppia di parentesi)

Aggiungiamo una coppia chiave-valore alla QUERY_STRING:
HisId=\$HisId

Vogliamo ora trasformare ogni riga da questo formato:

201 (1) Titolo dell'istogramma

a questo formato (con uno hyperlink)

(\s+ (\d+) \s+ \(\d\) \s+)(.*)

201 (1) [Titolo dell'istogramma](#)

\$1

\$2

\$1 \$2



Netscape: Esercizio per il Corso PERL della CNTC

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: <http://inf-n Corsi15.corsi.infn.it/~user15/> What's Related

Version 1.26/04 of HIGZ started
 *** No default PAWLOGON file "/tmp/.pawlogon.kumac" found
 File: /pool/user15/public_html/data/histograms.his

List of existing files

```

  ===> Directory :
    201 (2)  NR
    1201 (2) Phases NR
    202 (2)  <\[r\]\(770\)
    1202 (2) Phases <\[r\]\(770\)
    203 (2)  f?2!\(1275\)
    1203 (2) Phases f?2!\(1275\)
    204 (2)  f?0!\(980\)
    1204 (2) Phases f?0!\(980\)
    205 (2)  s?0!\(1475\)
    1205 (2) Phases s?0!\(1475\)
    206 (2)  <\[r\]\(1450\)
    1206 (2) Phases <\[r\]\(1450\)
    207 (2)  f?0!\(400\)
    1207 (2) Phases f?0!\(400\)
    208 (2)  f?0!\(1300\)
    1208 (2) Phases f?0!\(1300\)
    401 (2)  1
    402 (2)  2
    8000 (1)  Input file: master fitter generic ds ppp.his output file: ppp effi ds.his
    1313 (1)  Dalitz Dati
    1314 (1)  Dalitz Dati
    1510 (2)  Dalitz Dati
    1514 (2)  Dalitz Effi Check
    2001 (1)  <\[PPP\] mass \(Montecarlo\)
    1503 (2)  ds side bands
    1501 (2)  ds model
    1000 (1)  Full parameters space description
    3201 (1)  NR
    3202 (1)  <\[r\]\(770\)
    3203 (1)  f?2!\(1275\)
    3204 (1)  f?0!\(980\)
    3205 (1)  s?0!\(1475\)
    3206 (1)  <\[r\]\(1450\)
    3207 (1)  f?0!\(400\)
    3208 (1)  f?0!\(1300\)
    1515 (2)  Dalitz Adaptive Binning
    1505 (1)  Initial MinIMC Parameters
    1506 (1)  Fit MinIMC Parameters
    1507 (1)  Fit Informations
    1504 (2)  Fit Dalitz function \(S+B\)
    1519 (2)  Fit Dalitz function \(S\)
    1516 (2)  New background function from full fit
  
```

Lo script adesso produce questo formato dotato di opportuni hyperlinks

Prima di provarne la funzionalità dobbiamo implementare una funzione che risponda alla nostra richiesta (per esempio con una semplice scritta dimostrativa)...



```

sub ShowHistograms {
    &CreateKumacFile() ;

    print("<PRE>\n") ;

    $ENV{HOME} = "/tmp" ;

    open( PAW, "/cern/pro/bin/pawX11 -b $KumacFile |" ) ;
    while (<PAW>) {
        if ( m/^\s+(\d+)\s+(\d+)\s+(.+)/ ) {
            $HisId = $1 ;
            $Link = "http://${ServerMachine}/${ServerScript}" . "?" .
                "Action=PrintHistogram" . "&" .
                "HisId=$HisId" . "&" .
                "File=$File" ;
            s|^\(\s+\d+\s+(\d+)\s+(\d+)\s+(.+) (.*)|$1<A HREF="$Link">$2</A>| ;
        }
        print ;
    }
    close(PAW) ;
}

```

```

&MakeHeader() ;
&Initialize() ;
&DecodeQueryString() ;

```

```

if ($Action =~ m/ShowHistograms/) {
    &ShowHistograms() ;
} elsif ($Action =~ m/PrintHistogram/) {
    &PrintHistogram() ;
} else {
    &ShowFileList() ;
}

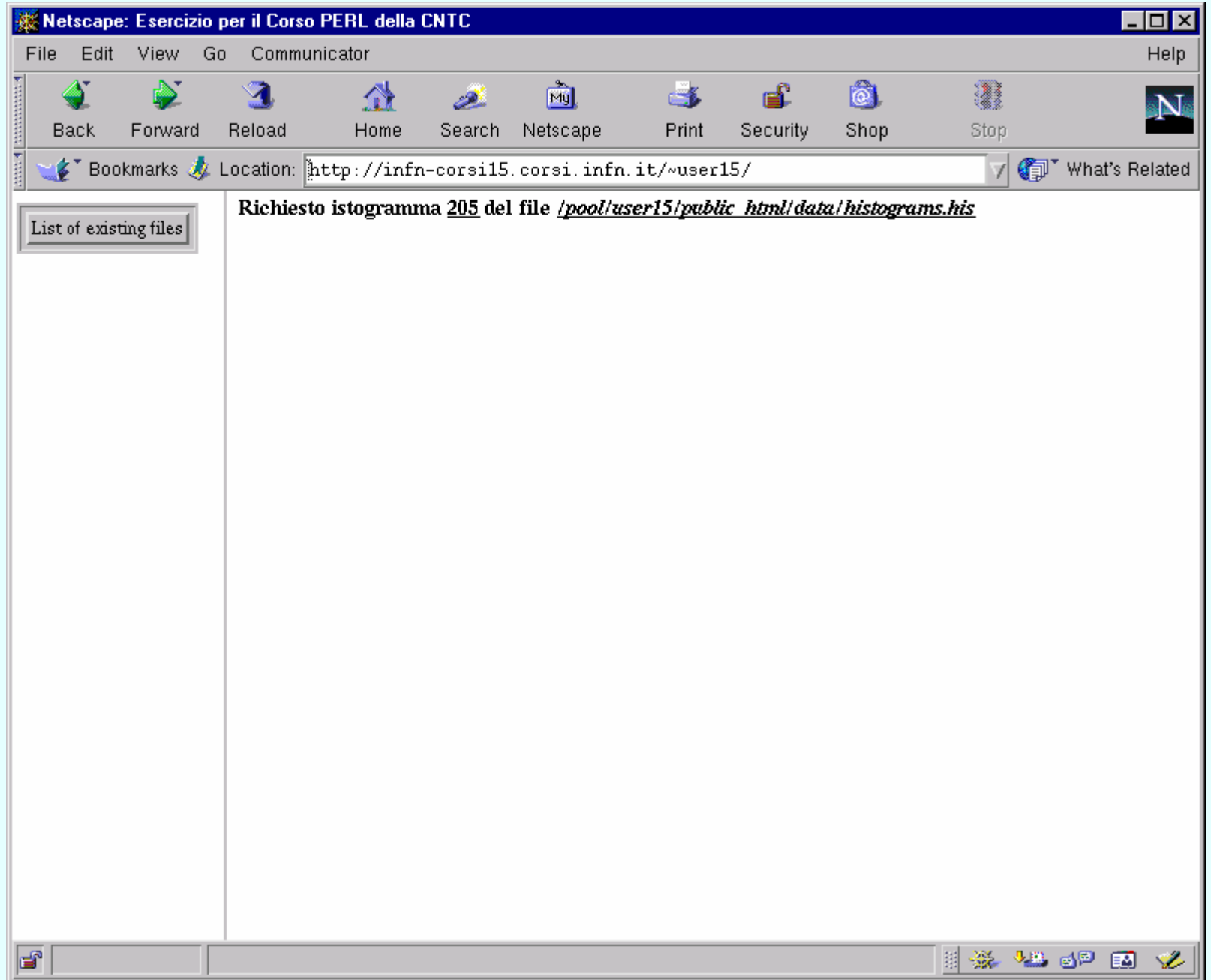
```

```

sub PrintHistogram {
    print <<EOT ;
    <H3>
    Richiesto istogramma <U>$HisId</U> del file <U><I>$File</I></U>
    </H3>
    EOT
}

```





```

sub PrintHistogram {
    open( KUM, ">${KumacFile}" );
    print KUM<<End_Of_Text ;
Macro ShowList

    His/File 1 $File
    H/Print $HisId

End_Of_Text

    close(KUM) ;

    $ENV{HOME} = "/tmp" ;
    print ("<PRE>\n") ;

    open( PAW, "/cern/pro/bin/pawX11 -b ${KumacFile} |" ) ;
    while (<PAW>) {
        print ;
    }
    close(PAW) ;
}

```

Il nuovo script **KUIP** conterrà ora un comando per la stampa dell'istogramma (sempre sullo **STDOUT**)

Un dettaglio tecnico, sottile, ma importante: **PAW** pretende di scrivere nella zona **HOME** dell'utente il file **last.kumac**. Per fare ciò usa la variabile ambientale **\$HOME**, che però, nel caso dell'utente *nobody* non è definita. Rimediamo a questa carenza definendola noi mediante l'uso della **HASH** implicita **ENV**.




```
$PostscriptFile = "/tmp/picture_${HisId}.ps" ;  
$GifFile = "/tmp/picture_${HisId}.gif" ;
```

```
open( KUM, ">${KumacFile}" ) ;  
print KUM<<End_Of_Text ;
```

Macro ShowList

```
His/File 1 $File  
H/Print $HisId
```

```
For/Fil 88 $PostscriptFile UNKNOWN  
Meta 88 -111
```

```
Set Ncol 56  
Palette 1  
Zon 2 2
```

```
H/Plot $HisId ColZ  
H/Plot $HisId surf4
```

```
Prox $HisId  
Proy $HisId  
H/Proj $HisId
```

```
H/Plo ${HisId}.prox  
H/Plo ${HisId}.proy
```

```
Close 88
```

End_Of_Text

```
close(KUM) ;
```

Arricchiamo lo script **KUIP** di tutti i pezzi di codice che permettono la generazione di istogrammi e facciamo in modo che **PAW** scriva un file di tipo **POSTSCRIPT** con quattro istogrammi (2x2)

Creiamo un hyperlink al file **GIF** (che dobbiamo ancora creare).

```
print <<EOT ;  
<PRE>  
<H2>  
<A HREF="http://$ServerMachine/~user15/$GifFile">GIF version</A>  
</H2>  
EOT
```

```
system("rm $PostscriptFile") ;  
open( PAW, "/cern/pro/bin/pawX11 -b $KumacFile |" ) ;  
while (<PAW> ) {  
  print ;  
}  
close(PAW) ;  
  
$Convert = "/usr/bin/X11/convert -density 80 -crop 0x0 " ;  
system("rm $GifFile") ;  
system("$Convert $PostscriptFile $GifFile" ) ;
```

Utilizziamo la utility di **UNIX** *convert* per convertire un file **POSTSCRIPT** in un file **GIF**



Netscape: Esercizio per il Corso PERL della CNTC

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

Bookmarks Location: <http://inf-n-corsi15.corsi.infn.it/~user15/> What's Related

List of existing files

```

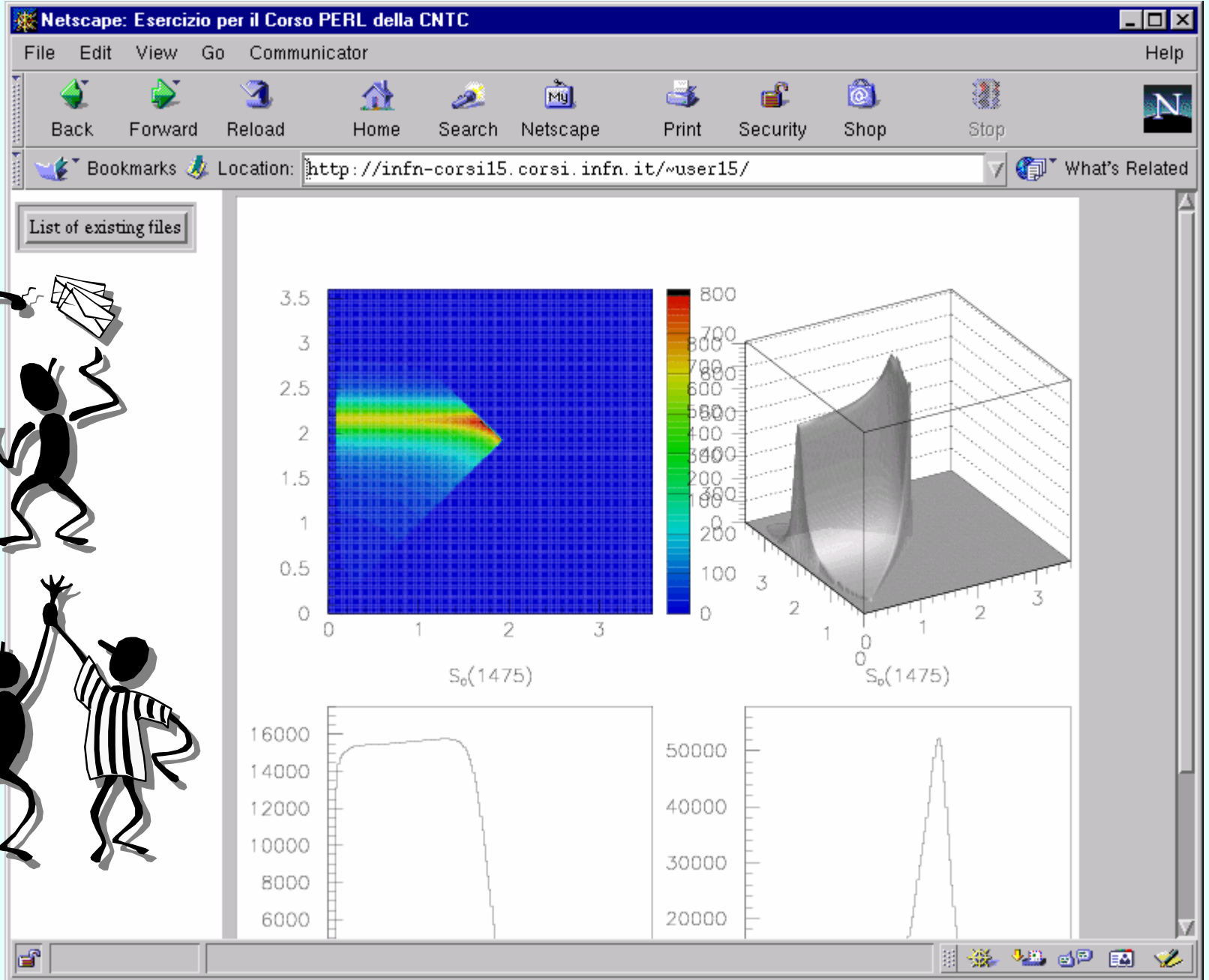
Version 1.26/04 of HIGZ started
*** No default PAWLOGON file "/tmp/.pawlogon.kumac" found

File: /pool/user15/public_html/data/histograms.his

====> Directory :
201 (2) NR
1201 (2) Phases NR
202 (2) <\[r\]\(770\)
1202 (2) Phases <\[r\]\(770\)
203 (2) f?2!\(1275\)
1203 (2) Phases f?2!\(1275\)
204 (2) f?0!\(980\)
1204 (2) Phases f?0!\(980\)
205 (2) s?0!\(1475\)
1205 (2) Phases s?0!\(1475\)
206 (2) <\[r\]\(1450\)
1206 (2) Phases <\[r\]\(1450\)
207 (2) f?0!\(400\)
1207 (2) Phases f?0!\(400\)
208 (2) f?0!\(1300\)
1208 (2) Phases f?0!\(1300\)
401 (2) 1
402 (2) 2
8000 (1) Input file: master fitter genric ds ppp.his output file: ppp effi ds.his
1313 (1) Dalitz Dati
1314 (1) Dalitz Dati
1510 (2) Dalitz Dati
1514 (2) Dalitz Effi Check
2001 (1) <\[PPP\] mass \(Montecarlo\)
1503 (2) ds side bands
1501 (2) ds model
1000 (1) Full parameters space description
3201 (1) NR
3202 (1) <\[r\]\(770\)
3203 (1) f?2!\(1275\)
3204 (1) f?0!\(980\)
3205 (1) s?0!\(1475\)
3206 (1) <\[r\]\(1450\)
3207 (1) f?0!\(400\)
3208 (1) f?0!\(1300\)
1515 (2) Dalitz Adaptive Binning
1505 (1) Initial MinimC Parameters
1506 (1) Fit MinimC Parameters
1507 (1) Fit Informations
1504 (2) Fit Dalitz function \(S+B\)
1519 (2) Fit Dalitz function \(S\)
1516 (2) Pure background function from full fit

```





L'estrema parsimonia sintattica di PERL puo' essere utilizzata con grande vantaggio, purché cio' sia fatto *cum grano salis*, in quanto il risultato puo' essere qualcosa di estremamente incomprensibile e quindi di difficile manutenzione.

Un paio di esempi tratti dallo *Annual Obfuscated PERL Contest*:

```
#!/usr/bin/perl -s

sub R{int$_[0]||
return vec$_[1],$_[2]/4,32;int$_[0]*rand}($R)
=$^=~'([\]-\`]' ;sub F{$u=0;grep$u!=$S->[_][$_[0]>>
$_*4&15]<<$*4,reverse 0..7;$u<<11|$u>>21}$t=$e
|_|$d?join' ',<>:({$p,$d}=(R,1),unpack u
,(3=MCV7%2W'<'");@b=@t=0..15;for(
;$i<length$p;$i+=4){srand($s^=R$R,$p
,$i)}while($c<8){grep{push@b ,splice
@b,R(9),5}@t;$R[$c]=R(2 **32);@{
$S->[$c++]}=@b}@h=0..7;@o =reverse
@h;while($a<length
$t){$v=R$R,$t,$a;
$w=R$R,$t,($a+=8)-4;
grep$q++%2?$v
^=F$w+$R
[$$R]:(
x3):((
print
$w^=F$w+$R[$$R]),$d?(@h,(@o
@h)x3,@o);$_.=pack N2,$w,$v}
```

Molto bello esteticamente, ma francamente non sono riuscito a capire cosa fa...

```

        $n="0m26<1~1 *,..VztRE@<620,*-! ";$B
        =65521;use integer;$sb='{ $a=shift;'.
        '$g=$h=$i=0;@c=';eval "sub u".$sb.'map{'
        '$g+=$_*$a;$b=$g%$B;$g/=$B;$b;}@_!$g||'.
    'push@c      , $g;@c;}'      ;eval"sub"
    . 'd'          . $sb.          'reverse'
    . 'map         { $h=(          $_+$g*$B'
    . ')/          $a;$g          =($_+$g*'
    . '$B)        -$h*$          a;$h;}re'
    .             . 've'          . 'rse@_';
    .             . '@c[         $#c]!=0|'
    .             . '|p'         . 'op@c;('
    .             . '$g'         . ',@c);}'
    ;sub p{for($j=0          ;(($k,@l)
    =d(2,@_))&&$k          ==0;$j++) {@_=@1;}
return$j*("@"_         eq 1)&&($#_==0
);}push@o,2;          do{($r,@t)=d(
ord(substr$n         , $m+14,1)-31,
@o);if($r==         0){@o=u(ord(
substr$n, $m,         1)-31,@t);$m
=0;if($p=           p(@o){print
"$p\n";}}          else{$m=($m
+1)%14;}            }while($n);

```

Calcola e stampa i numeri primi... (sic)

Documentazione
in rete: la home page
<http://www.perl.com>

www.perl.com
O'REILLY

Perl Versions: [Stable is 5.6.0](#), [Devel is 5.7.0](#)

binaries

THE SOURCE FOR PERL
www.perl.com

DOWNLOADS
DOCUMENTATION
CPAN
FAQs
TRAINING
MARKETPLACE
FEATURES
REFERENCE
SEARCH

What is Perl?

[Where can I find the latest version?](#)

[Is there a Windows version of Perl?](#)

[How can I learn to write CGI scripts?](#)

P5P Digest
[This Week on p5p 2001/02/12](#)
Perl FAQ updates, memory leak plumbing, and more. [P5P Digest Archive](#)

Features

[Perl 6 Alive and Well! Introducing the perl6-mailing-lists Digest](#)
Perl.com will be supplying you with the P6P digest, covering the latest news on the development of Perl 6.

[Pathologically Polluting Perl](#)
Brian Ingerson introduces Inline.pm and CPR; with them you can embed C inside Perl and turn C into a scripting language.

[Quick Start with SOAP](#)
An introduction to SOAP::Lite, a module that provides simple yet flexible interface to SOAP, a popular XML-RPC protocol. Using SOAP::Lite, Perl scripts can access objects and execute procedures on remote servers, and also serve SOAP objects and procedures over the Net.

What's New?

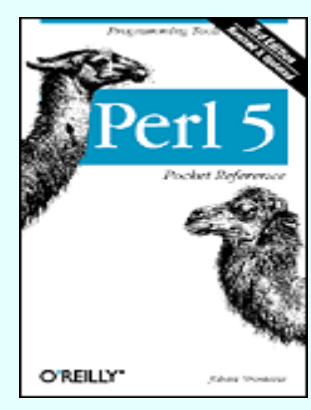
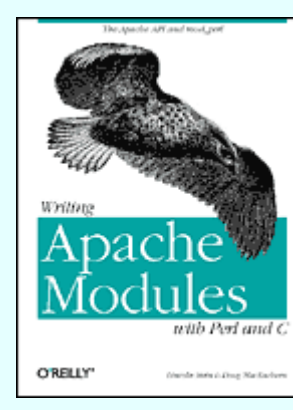
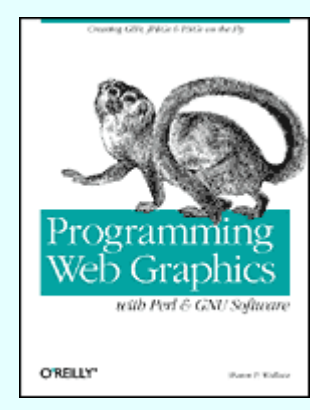
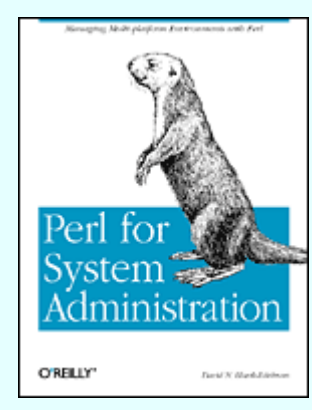
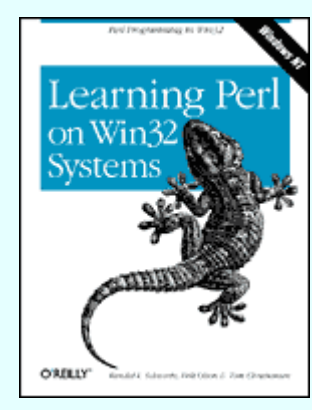
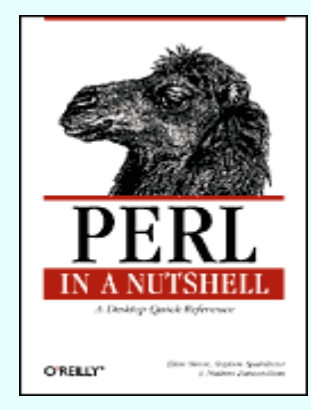
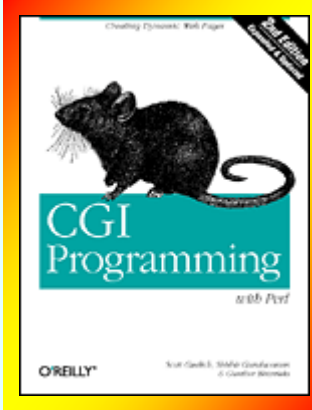
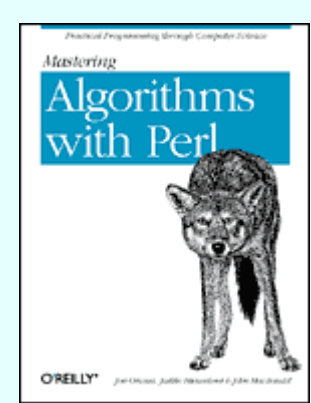
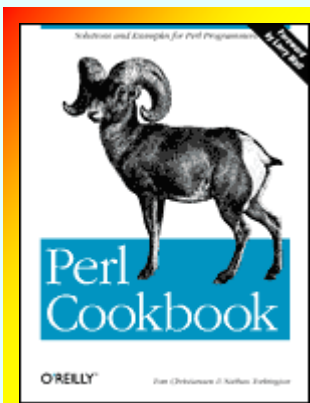
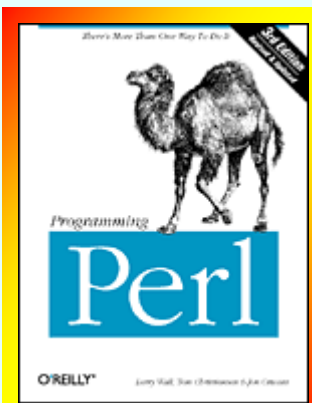
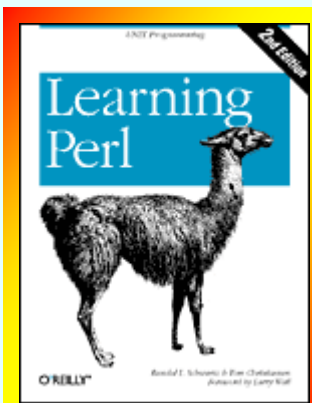
- [Camel Made of Code T-Shirt](#)
- [3rd German Perl Workshop](#)
- [Perl Conference 5 Call for Participation](#)

[More News...](#)

Success Stories

[NBCi Accesses and Maintains Complex Databases with Perl](#)
-- This month's Perl Success Story tells how NBCi wrangles an enormous amount of information with Perl by





Comprehensive Perl Archive Network - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Stop

Bookmarks Location: <http://www.cpan.org/> What's Related

My own home pag Focus home page Istituto Nazion Yahoo The Java Tutori RealPlayer

CPAN

Comprehensive Perl Archive Network

Welcome to CPAN! Here you will find All Things Perl.

CPAN is the **Comprehensive Perl Archive Network**. *Comprehensive*: the aim is to contain all the Perl material you will need. *Archive*: 749 megabytes as of January 2001. *Network*: CPAN is mirrored at more than one hundred [sites](#) around the world.

Browsing the Archive

- [documentation](#)
 - [Randy Kobes' Perl documentation server](#)
 - [Carlos Ramirez' Perl documentation server](#)
- [modules](#)
- [scripts](#)
- [binary distributions \("ports"\)](#)
- [source code](#)
- [comp.lang.perl.announce archives](#)
- [recent arrivals](#)
- [recent](#) modules listed at CPAN Search

Searching the Archive

- [Perl core and CPAN modules documentation](#) (Randy Kobes)
- [Perl core documentation](#) (Carlos Ramirez)
- [CPAN modules, distributions, and authors](#) (search.cpan.org)
- [CPAN modules documentation](#) (The Perl Foundation)

Document: Done



Index of /modules/by-category - Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Stop

Bookmarks Location: <http://cpan.valueclick.com/modules/by-category/> What's Related

My own home pag Focus home page Istituto Nazion Yahoo The Java Tutori RealPlayer

Index of /modules/by-category

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory	10-Dec-2000 05:13	-	
02 Perl Core Modules/	14-Nov-2000 17:06	-	
03 Development Support/	07-Dec-2000 04:06	-	
04 Operating System Interfaces/	11-Jan-2001 06:06	-	
05 Networking Devices IPC/	28-Jun-2000 21:08	-	
06 Data Type Utilities/	11-Nov-2000 02:06	-	
07 Database Interfaces/	07-Nov-2000	18 Images Pixmaps Bitmaps/	12-Nov-2000 06:06 -
08 User Interfaces/	18-Nov-2000	19 Mail and Usenet News/	20-Oct-2000 04:08 -
09 Language Interfaces/	23-May-2000	20 Control Flow Utilities/	26-Jun-1998 08:28 -
10 File Names Systems Locking/	21-May-2000	21 File Handle Input Output/	22-Mar-2000 22:59 -
11 String Lang Text Proc/	19-Nov-2000	22 Microsoft Windows Modules/	30-Dec-1997 12:49 -
12 Opt Arg Param Proc/	02-Apr-2000	23 Miscellaneous Modules/	06-Nov-2000 23:08 -
13 Internationalization Locale/	12-Jun-2000	24 Commercial Software Interfaces/	18-Apr-2000 19:44 -
14 Security and Encryption/	05-Apr-2000	99 Not In Modulelist/	29-Jul-1999 20:27 -
15 World Wide Web HTML HTTP CGI/	11-Jan-2001 13:06	-	
16 Server and Daemon Utilities/	03-Feb-1999 17:10	-	
17 Archiving and Compression/	12-Oct-2000 03:08	-	

Document: Done



Fine del corso

