Università Roma Tre – Facoltà di Scienze M.F.N. – Corso di Laurea in Matematica

Corso di Informatica Generale 1 – IN1

Linguaggio Perl

Marco Liverani
(liverani@mat.uniroma3.it)

Sommario

- Prima parte: *caratteristiche del linguaggio*
  - Caratteristiche e obiettivi generali
  - Sintassi
  - Strutture dati
  - Operazioni di input/output
  - Operatori e strutture di controllo
  - Espressioni regolari e pattern-matching
  - Subroutine e programmi esterni
- Seconda parte: *Perl per le applicazioni Web*
  - Server HTTP e interfaccia CGI
  - Struttura di una applicazione CGI
  - Esempi

Prima parte

Caratteristiche del linguaggio Perl

(*Practical Extraction and Report Language*)

Obiettivi generali

- Perl nasce nel 1987 come linguaggio di *scripting* per automatizzare operazioni ripetitive per la gestione di sistemi UNIX
- Viene progettato quindi come uno strumento agile mediante cui si potesse realizzare rapidamente piccoli programmi per
  - L'elaborazione di file ASCII (file di configurazione, file di log, file di dati, …)
  - L'elaborazione di stringhe
- Operazioni simili possono essere realizzate con i comandi della shell e con i programmi di base di ogni sistema UNIX (grep, sed, awk, …), ma Perl fornisce un ambiente (un linguaggio) più "confortevole"

Caratteristiche

- Perl è un linguaggio interpretato: l'interprete del linguaggio è disponibile per numerosissime piattaforme
- Il linguaggio Perl non è "tipato": il tipo di dato assegnato ad una variabile non deve essere dichiarato, l'interprete lo deduce dal contesto
- Perl è un linguaggio imperativo procedurale di alto livello, non consente un accesso diretto alla memoria della macchina
- Perl è anche un linguaggio ad oggetti: è possibile definire classi e metodi

Dicembre 2002 M. Liverani - Linguaggio Perl 5

Aspetti rilevanti

- Il linguaggio Perl mette a disposizione del programmatore alcuni strumenti molto potenti, non disponibili in modo così diretto in altri linguaggi:
  - Trattamento di file sequenziali molto naturale ed immediato
  - Manipolazione delle stringhe anche mediante le espressioni regolari
  - Gestione dinamica di array e liste
  - Gestione dinamica di array associativi (*hash table*)

Dicembre 2002 M. Liverani - Linguaggio Perl 6

Sintassi

- La sintassi del linguaggio Perl è mutuata da quella del linguaggio C (es.: ogni istruzione termina con un punto-e-virgola, le strutture di controllo per i cicli sono simili, ecc.)
- L'esecuzione del programma inizia dalla prima istruzione "utile" (non esiste una funzione *main*)
- Il programma può essere suddiviso in più subroutine, anche ricorsive, che possono restituire o meno un valore
- Ogni subroutine ammette un numero di argomenti variabile
- Il passaggio degli argomenti alle subroutine avviene sempre per valore
- I programmi Perl in ambiente UNIX iniziano con una riga con il riferimento al programma interprete:

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


Strutture dati

- Il Perl gestisce direttamente diverse strutture dati:
  - Variabili scalari (es.: `$a`, `$nome`, …)
  - Liste o *array* (es.: `@a`, `@studenti`, …)
  - Array associativi o *hash table* (es.: `%a`, `%nome`, …)
- Liste ed array sono gestiti nello stesso modo:
  - gli array non hanno una dimensione prefissata
  - gli elementi di una lista possono essere individuati tramite un indice intero
- L'espressione `$#`*nome_lista* fornisce l'indice dell'ultimo elemento di un array (il primo elemento ha indice 0)
- Un singolo elemento di una lista o di un hash table è una variabile scalare (es.: `$a[3]`, `$studenti[$i]`, `$matrice[$i][$j]`, `$nome{'liverani'}`, …)

Operatori

- Operatori aritmetici: "`+`" somma, "`-`" sottrazione, "`*`" prodotto, "`/`" divisione, "`%`" modulo (resto della divisione intera)
- Operatore di assegnazione (in forma estesa): "`=`" (es.: "`$a = $b*$c;`", "`$x = -27.4;`")
- Operatori aritmetici di assegnazione (in forma compatta): "`++`", "`--`" (es.: "`$x--;`" decrementa di uno il valore di `$x`, è equivalente a "`$x=$x-1;`", ma è più efficiente), "`+=`", "`-=`", "`*=`", "`/=`" (es.: "`$x += 7;`" è equivalente a "`$x = $x+7;`")

Dicembre 2002 M. Liverani - Linguaggio Perl 9

Operatori logici e booleani

- Operatori logici di confronto

Operatore	Numeri	Stringhe
Uguale a	`$a == $b`	`$a eq $b`
Diverso da	`$a != $b`	`$a ne $b`
Maggiore	`$a > $b`	`$a gt $b`
Maggiore o uguale	`$a >= $b`	`$a ge $b`
Minore	`$a < $b`	`$a lt $b`
Minore o uguale	`$a <= $b`	`$a le $b`

- Operatori booleani: and = "`&&`", or ="`||`", not="`not`"
- Le espressioni logiche vengono valutate da sinistra verso destra; la valutazione di una espressione termina non appena è possibile stabilirne con certezza il valore

Dicembre 2002 M. Liverani - Linguaggio Perl 10

Input/Output su file

- Tutte le operazioni di I/O il Perl le esegue da e verso file
- Ogni programma ha a disposizione tre file sempre aperti:
  - STDIN: STanDard INput, corrisponde tipicamente alla tastiera del terminale dell'utente
  - STDOUT: STanDard OUTput, corrisponde tipicamente al video del terminale dell'utente
  - STDERR: STanDard ERRor, corrisponde al *device* di output su cui devono essere visualizzati gli errori
- Altri file possono essere aperti mediante l'istruzione open:
  - `open filein, "< dati.txt";` (aperto in lettura)
  - `open fileout, "> dati.txt";` (aperto in scrittura)
  - `open fileout, ">> dati.txt";` (aperto in *append*)
- I file possono essere chiusi con l'istruzione **`close`**:
  - `close filein;`

Dicembre 2002 M. Liverani - Linguaggio Perl 11

Esempi di I/O

```
#!/usr/bin/perl
# Programma di esempio
print "Nome del file:";
$nome = <STDIN>;
open dati, "< $nome";
$n = <dati>;
@a = <dati>;
print "n = $n \n";
print "@a \n";
close dati;
```

Interprete del linguaggio

commento

Lettura di una riga da tastiera

Lettura di una riga da file

Legge il file fino alla fine e memorizza le righe nelle celle dell'array `@a`

Stampa un array per intero

Dicembre 2002 M. Liverani - Linguaggio Perl 12

Subroutine

- Le subroutine vengono dichiarate attraverso l'istruzione **sub** seguita dal nome della subroutine
- Il blocco delle istruzioni di una subroutine è delimitato dalle parentesi graffe
- La subroutine viene richiamata con il suo nome preceduto da una "**&**" e seguito dai parametri passati come argomento
- Il numero di parametri di una subroutine è variabile: i parametri vengono raccolti in una lista e assegnati alle variabili attraverso l'istruzione **shift**
- In una subroutine possono essere definite variabili locali con l'istruzione **my**
- La subroutine termina con l'istruzione **return** che consente di specificare il valore (o i valori) restituiti dalla subroutine

Esempio di subroutine

```
#!/usr/bin/perl
# Media aritmetica
sub media {
  my ($x, $y, $z, $M);
  $x = shift;
  $y = shift;
  $z = shift;
  $M = ($x+$y+$z)/3;
  return($M);
}
print "Inserisci 3 numeri:";
chop($a = <STDIN>);
chop($b = <STDIN>);
chop($c = <STDIN>);
$m = &media($a, $b, $c);
print "La media tra $a, $b e $c e' $m\n";
```

Definizione di una subroutine

Dichiarazione di variabili locali

Assegnazione dei parametri della funzione alle variabili

Valore restituito dalla subroutine

Chiamata di una subroutine

La struttura condizionale

- Il Perl favorisce la programmazione strutturata attraverso alcune strutture di controllo
- La struttura condizionale "*se … allora … altrimenti …*" ha diverse forme
- Forma canonica (analoga al linguaggio C):

```
if (condizione) { blocco istruzioni }
else { altre istruzioni }
```

- Forma compatta:

```
(condizione) && (istruzione);
(condizione) || (istruzione);
```

- Forma inversa:

```
unless (condizione) { blocco di istruzioni }
```

Dicembre 2002 M. Liverani - Linguaggio Perl 15

Strutture iterative

- Esistono diverse strutture di controllo per l'implementazione di cicli:
- `while (condizione) { blocco di istruzioni }`

  Esempio: `while ($i<$n) {…}`
- `for (ass. iniziale; cond. finale; incremento) { blocco di istruzioni }`

  Esempio: `for ($i=0; $i<=$#A; $i++) { … }`
- `foreach variabile (lista) {blocco di istruzioni}`

  Esempi:

```
foreach $a (@b) { … }
foreach $k (sort keys %a) { … }
```

Dicembre 2002 M. Liverani - Linguaggio Perl 16

Esempio

- Legge in input il contenuto di un file e lo riscrive ordinato:

```
#!/usr/bin/perl
# Ordinamento di un file
chop($file = shift);
open IN, "< $file" || die "Errore\n";
@dati = <IN>;
close IN;
@dati = sort(@dati);
open OUT, "> $file" || die "Errore\n";
foreach $k (@dati) {
  print OUT "$k";
}
close OUT;
```

Ordina una lista e restituisce la lista ordinata

Termina il programma

Dicembre 2002 M. Liverani - Linguaggio Perl 17

Espressioni regolari

- Perl consente di definire delle espressioni per rappresentare classi di stringhe attraverso le espressioni regolari che ne descrivono lo "schema"
- Sono costruite componendo alcuni simboli che svolgono il ruolo di meta caratteri :
  `.` = qualsiasi carattere `\s` = un simbolo di spaziatura
  `\d` = una cifra `\w` = una lettera
- Altri simboli servono come quantificatori :
  `?` = zero o una volta, `+` = una o più volte, `*` = zero o più volte
- da ancore che forzano la posizione della sottostringa :
  `^` = la stringa inizia per… `$` = la stringa termina per…
- Altri caratteri rappresentano se stessi : `a`, `b`, `c`, `1`, `2`, `3`, ecc.

Dicembre 2002 M. Liverani - Linguaggio Perl 18

Espressioni regolari – esempi

- `\d+\s\w*`

  almeno un carattere numerico, uno spazio, un numero arbitrario di caratteri qualsiasi

  Esempi: "`22 abcdKKaa`", "`1 `", "`1 a`", "`xyz123 abcd22 456efg`"

- `^Nome:.+\sEtà:\d+$`

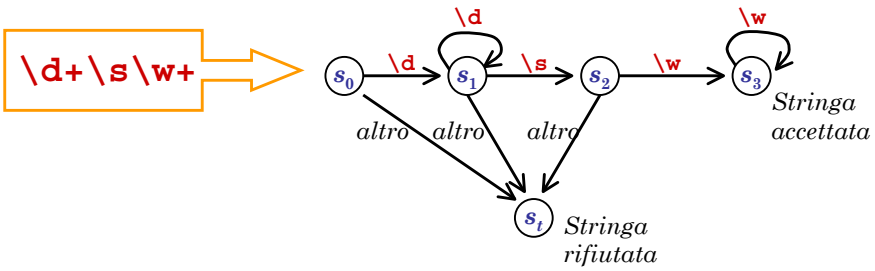
  una riga che inizia con la stringa "**`Nome:`**", seguita da una stringa non vuota di caratteri qualsiasi, uno spazio, la sottostringa "**`Età:`**" e che termina con un numero

  Esempi: "`Nome: Giovanni Età: 22`"

Dicembre 2002 M. Liverani - Linguaggio Perl 19

Espressioni regolari e automi riconoscitori

- Le espressioni regolari descrivono un automa riconoscitore, in grado di accettare o rifiutare determinate stringhe in base alla grammatica specificata con l'espressione regolare



Dicembre 2002 M. Liverani - Linguaggio Perl 20

Pattern matching

- Consiste nel verificare che una stringa o una sua sottostringa sia costruita secondo uno schema descritto da una espressione regolare:
- L'istruzione

  $var **=~** **/***espressione regolare*/[**i**]

  restituisce "vero" se il pattern matching riesce, "falso" altrimenti.
- Esempi :

```
if ($testo =~ /^Subject:\s.*$/i) {
  print "$testo\n";
}

while ($riga = <IN>) {
  $riga =~ /^Subject:\s(.*)$/ && print "$1\n";
}
```

Dicembre 2002 M. Liverani - Linguaggio Perl 21

Pattern substitution

- Sostituzione di sottostringhe individuate mediante espressioni regolari con altre stringhe
- L'istruzione

  $*var* **=~** **s/***espr. reg.*/*stringa di sostituz.*/**[g] [i]**

  sostituisce in **$***var* la sottostringa descritta dall'espressione regolare con la stringa di sostituzione
- Con le parentesi si memorizzano sottostringhe, da richiamare in seguito con **\1**, **\2**, ... e **$1**, **$2**, ...
- Esempi:
  - `$var =~ s/Nome:\s(\w+)\sEtà:\s(\d+)/\1, \2 anni/;`
  -
```
for ($i=0; $i<=$#array; $i++) {
  $array[$i] =~ s/$v1/$v2/g;
}
```
  - `$data =~ s/(\d\d)\/(\d\d)\/(\d\d)/\1 $mese[\2] \3/;`

Dicembre 2002 M. Liverani - Linguaggio Perl 22

Apici, doppi apici, backtick

- In Perl gli apici hanno un significato ben preciso, ma spesso differente da quello adottato in altri linguaggi:
  - L'apice singolo «**'**» serve a delimitare le stringhe; es.:
    ```
    $a = 'Marco Liverani';
    $b = 'Valore: 1000$'
    ```
  - L'apice doppio «**"**» serve per delimitare stringhe consentendo "l'interpolazione": l'interprete sostituisce ai nomi delle variabili contenute nella stringa il loro valore; es.: `$a = "Nome: $nominativo\n"`
  - L'apice "inverso" «**`**» (in gergo *backtick*) è un operatore che consente di eseguire comandi esterni al programma, catturare l'output e restituirlo al programma Perl; es.:
    ```
    $data = `date +%d/%m/%Y`
    @nomi = `grep Marco nominativi.txt | sort`
    ```


Seconda parte

Perl per le applicazioni CGI

(*Common Gateway Interface*)

Programmi CGI

- CGI, *Common Gateway Interface*, è la modalità con cui è possibile eseguire dei programmi su un server Web
- L'interfaccia CGI definisce il modo in cui è possibile inviare delle informazioni (parametri) da un client web (un web browser) all'applicazione invocata sul server web e come sia possibile restituire al client l'output prodotto dal programma
- I programmi che operano secondo tale modalità vengono definiti programmi CGI
- I programmi CGI vengono eseguiti sul server, non sul client
- Il Perl è uno dei principali strumenti per lo sviluppo di applicazioni CGI

Dicembre 2002 M. Liverani - Linguaggio Perl 25

Protocollo HTTP

Dicembre 2002 M. Liverani - Linguaggio Perl 26

Server Web (o server HTTP)

- Un server Web è un programma che implementa la componente server del protocollo HTTP
- Il server Web esegue questa sequenza di operazioni:
  1. Accetta la connessione da parte di un client mediante il protocollo HTTP
  2. Verifica se il client o l'utente ha l'autorizzazione per eseguire il collegamento
  3. Accetta una richiesta dal client mediante il protocollo HTTP
  4. Per richieste di risorse statiche (file HTML, immagine JPG, ecc.): il server carica la risorsa e la invia al client
  5. Per richieste di risorse dinamiche: il server esegue il programma passandogli attraverso la modalità CGI i parametri ricevuti dal client ed invia al client l'output prodotto dal programma
  6. Chiude la connessione con il client

Dicembre 2002 M. Liverani - Linguaggio Perl 27

Passaggio di parametri

- I parametri vengono forniti con un'unica stringa in formato URL encoded:

  $nome_1$=$valore_1$&$nome_2$=$valore_2$&…&$nome_n$=$valore_n$

- Esempio:

  `nome=Silvia&cognome=Di+Stefano&eta=23`

- È il server HTTP che si occupa di passarli al programma CGI secondo due modalità:
  - POST: la stringa viene fornita sul canale standard input, il programma leggerà la stringa da STDIN
  - GET: la stringa viene memorizzata nella variabile di ambiente **`QUERY_STRING`**
- La modalità viene specificata dal server (sulla base delle indicazioni del client) mediante la variabile di ambiente **`REQUEST_METHOD`**

Dicembre 2002 M. Liverani - Linguaggio Perl 28

Compiti del programma CGI

- Acquisire la stringa con i parametri
- Eseguire il parsing dei parametri ricevuti in input, ossia una suddivisione della stringa per isolare le coppie "nome=valore" e successivamente per separare il nome del parametro dal suo valore
- Eseguire l'elaborazione richiesta
- Produrre un output compatibile con le capacità di visualizzazione del browser (tipicamente output in formato HTML)
- L'output deve contenere una intestazione che ne indichi il formato secondo la codifica MIME (*Multipurpose Internet Mail Extensions*); es.: "`Content-type: text/html`"

Programmi CGI in Perl

- In Perl esiste una libreria per implementare le principali funzioni della programmazione CGI: **`cgilib.pl`**
- Con l'istruzione **`require`** è possibile caricare una libreria nel programma Perl
- Es.: **`require "cgilib.pl";`**
- Tra le subroutine messe a disposizione dalla libreria c'è la "**`readparse`**":
  - Legge la stringa dei parametri
  - Suddivide i nomi dei parametri dai valori
  - Costruisce l'array associativo **`%in`**:

    `$in{`*nome***`}`** = *valore*

Esempio elementare

```
<html>
 <head><title>Input</title></head>
 <body>
  <form action="ciao.cgi" method="get">
  Come ti chiami? <input name="nome">
  <input type="submit">
  </form>
 </body>
</html>
```

```
#!/usr/bin/perl
# Programma di esempio

require "cgilib.pl";
&ReadParse;
print "Content-type: text/html\n\n";
print "<html><body>\n";
print "<p>Ciao $in{nome}</p>";
print "</body></html>\n";
```


Indirizzario on-line: schema dell'applicazione

- Programma Web per l'archiviazione di indirizzi e la ricerca sul catalogo.
- Sono necessari quattro file:
  - Form di input: pagina HTML con una form per l'inserimento dei dati da archiviare
  - Script di archiviazione: programma Perl che riceve i dati inseriti dall'utente e li archivia sul file
  - Form di query: pagina HTML per l'inserimento dei parametri con cui eseguire una ricerca in archivio
  - Script di ricerca: programma Perl per la ricerca dei record nell'archivio sulla base dei parametri specificati nella form precedente
- L'archivio è un file ASCII:
  - Ogni riga è un record
  - I campi sono separati fra loro da un simbolo "|" (*pipe*)

Indirizzario on-line: Form di input

```
<HTML>
 <HEAD>
  <TITLE>Input dei dati</TITLE>
 </HEAD>
 <BODY>
  <H1>Inserimento dati</H1>
  <FORM ACTION="/cgi-bin/salva.pl" METHOD="GET">
   <P>Nome: <INPUT NAME="nome"></P>
   <P>Cognome: <INPUT NAME="cognome"></P>
   <P>Citt&agrave;: <INPUT NAME="citta"></P>
   <P><INPUT TYPE="SUBMIT"></P>
  </FORM>
 </BODY>
</HTML>
```


Indirizzario on-line: Script di archiviazione

```
#!/bin/perl
require("/usr/local/lib/perl/cgilib.pl");
&ReadParse();
open(OUT, ">> archivio");
print OUT "$in{cognome}|$in{nome}|$in{citta}\n";
close(OUT);
print <<"FINE_TESTO";
Content-type: text/html

<HTML>
<BODY>
<P>Grazie per l'inserimento</P>
</BODY>
</HTML>
FINE_TESTO
```

Indirizzario on-line: Form di query

```
<HTML>
 <HEAD> <TITLE>Ricerca</TITLE> </HEAD>
 <BODY>
  <H1>Inserisci i parametri per la ricerca
   in archivio</H1>
  <FORM ACTION="/cgi-bin/cerca.pl"
   METHOD="POST">
   <P>Nome: <INPUT NAME="nome"> </P>
   <P><INPUT TYPE="SUBMIT"> </P>
  </FORM>
 </BODY>
</HTML>
```


Indirizzario on-line: Script di ricerca

```
#!/bin/perl
require("/usr/local/lib/perl/cgilib.pl");
&ReadParse;
print "Content-type: text/html\n\n";
print "<HTML><BODY>\n";
open (ARC, "< archivio");
while ($record=<ARC>) {
 ($cognome, $nome, $citta) = split(/\|/, $record);
 if ($nome =~ /^.*$in{nome}.*$/) {
  print "<P>Nome: $nome</P>";
  print "<P>Cognome: $cognome</P>";
  print "<P>Citt&agrave;: $citta</P>\n";
 }
}
close(ARC) ;
print "</BODY></HTML>\n";
```

Bibliografia

- L. Wall, *Programming Perl*, O'Reilly, 2000
- J. Orwant, J. Hietaniemi, J. Macdonald, *Mastering Algorithms with Perl*, O'Reilly, 1999
- S. Srinivasan, *Advanced Perl Programming*, O'Reilly, 1997
- Risorse disponibili su Internet:
  - http://www.perl.org
  - http://www.perl.com
  - http://cgi-lib.berkeley.edu/ (CGI-lib home page)
  - http://www.mat.uniroma3.it/users/liverani/perl