

An Introduction to Perl for bioinformatics

Will Hsiao

wwhsiao@sfu.ca

www.pathogenomics.sfu.ca/brinkman



Adapted from Sohrab Shah's original lecture, University of British Columbia Bioinformatics Centre (UBiC)

Lecture 8.1

An Introduction to Perl for bioinformatics

- Objective:
 - To demonstrate how Perl can be used in bioinformatics
 - To empower you with the basic knowledge and resources required to quickly and effectively create simple tools to process biological data
 - Write your own programs!
 - Give the programmers in the group a chance to help their biologist team-mates

Outline

- What is programming?
- What is Perl?
- Perl – a brief history
- Perl compared to other languages
- General Use of Perl
- Use of Perl in Bioinformatics
- A bit of code
- Lab preview

What is Programming?

- **Programs:** a set of instructions telling computer what to do
- **Programming languages:** bridges between human languages (A-Z) and machine languages (0&1)
- **Compilers** convert programming languages to machine languages

Machine
language



Human
language

Low Level Programming language: hard to write (more bugs), more flexible, runs faster

High Level Programming language: easier to write (fewer bugs), more rigid, runs slower

Assembly language C, C++

Java Perl, Shell languages,
VBASIC SQL

What is a program

Input: **data, parameters**



An Addition program:

Input: 5 and 3

BLASTP:

Input: "liinyplddqdaiveaact"
parameter: E-value cutoff

MS-Word:

Input: my thesis text, diagrams
parameter: save filename

Computer Programs

A black box for non-programmers

"Variables": used to hold a piece of information that can change with time ("tupperware of programming")

"Functions": Predefined actions that manipulate the variables and produce results

Output: results, files



An Addition program:

Output: 8

BLASTP:

Output: lac repressor

MS-Word:

Output: a formatted .doc file

Why Perl?

In Bioinformatics:

- A powerful tool for **quickly** automating analyses (it'll do BLAST 1,000,000 times for you happily)
- Sophisticated support and excellent performance for regular expression “**REGEX**” (it'll find all the ORFs (i.e. ATG...TAA) for you in a bacterial genome)
- Great support and large community (BioPerl, CPAN)

In this course:

- It is flexible and relatively easy to pick up – get it to work for you!
- It ties in well with what you have learned already (BLAST, UNIX)

What the /[^]\\$!\@|\%|*.* / is Perl?

- **P**ractical **E**xtraction and **R**eport **L**anguage
 - “PERL saved the human genome project” (Lincoln Stein)
- **P**athologically **E**clectic **R**ubbish **L**ister
 - “printer line noise”
- An interpreted programming language optimized for scanning text files and extracting information from them
- Fills in the gap between low level languages (C, assembly) and high level ones (shell languages)

A brief history in time

- Created by Larry Wall
- Perl 1.0 released in 1987
- Purpose: glue features of sed, awk, C, sh into a utility language that is flexible and easy to use
 - "In general, if you think something isn't in Perl, try it out, because it usually is. :-)"
 - "Historically speaking, the presence of wheels in Unix has never precluded their reinvention."
 - "Have the appropriate amount of fun."
 - "Let's say the docs present a simplified view of reality..."

A brief history in time (cont'd)

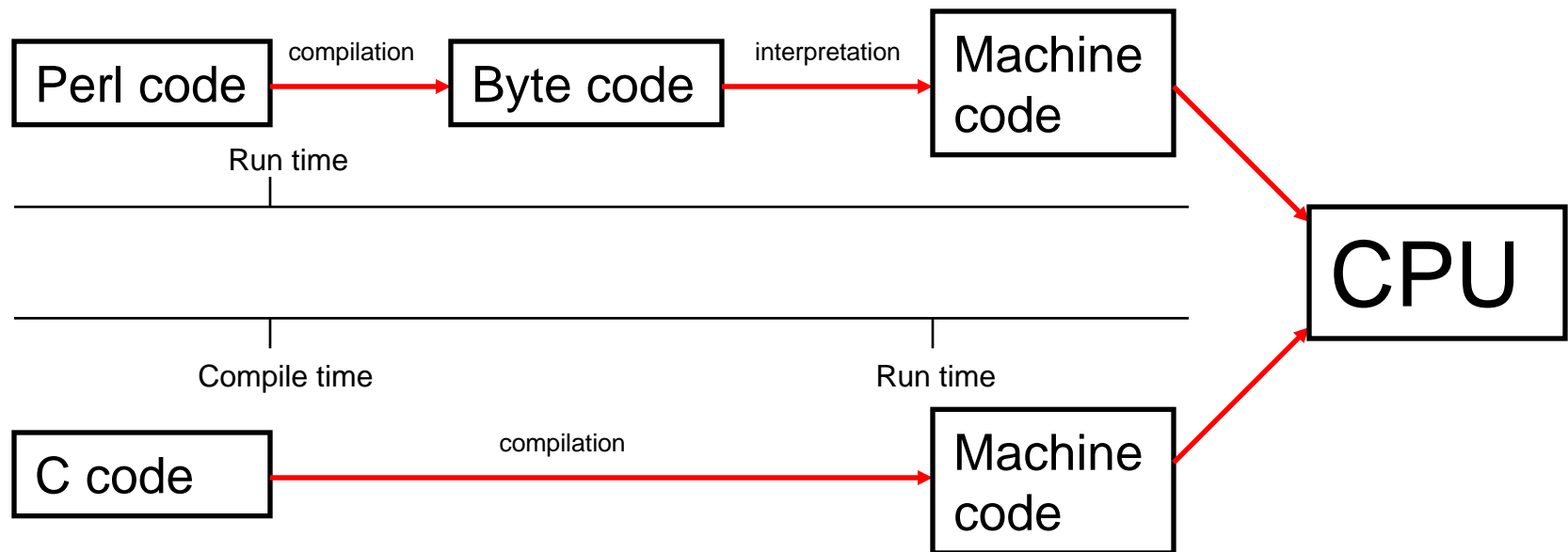
- 1989 – Perl released under the GPL
- 1991 – *Programming Perl* published by O'Reilly
- 1993 – CPAN conceived
- 1995 – Perl 5.000 released (objects)
 - first use of CGI
 - DBI module for Oracle
- 1996 – Perl journal published
- Now – Perl is everywhere

Source: history.perl.org/PerlTimeline.html

Perl Philosophy

- Interpreted → SLOW but more PORTABLE
 - Compiled into an intermediate byte code which is then interpreted
- Flexible – easy to learn for sed, awk, sh and C programmers
- Many useful built-in functions to make coding brief
- Object Oriented (sort of)
- A more “natural” language
 - words have different meanings in different contexts
- TMTOWTDI – There’s more than one way to do it
 - The Perl mantra
- Can do almost anything, anywhere

Perl is interpreted



Scripting languages are generally interpreted

Perl vs. the world

- Perl vs. C
 - C is a compiled language
 - C 'harder' to write and to port (e.g. Mac v.s. PC)
 - C faster to run, more memory efficient
 - Perl compiler/interpreter is written in C
- Perl vs. Python
 - Performance comparable
 - Python more elegant, more sophisticated, more readable
 - Lacks regex, file scanning, reporting features

Perl vs. the world

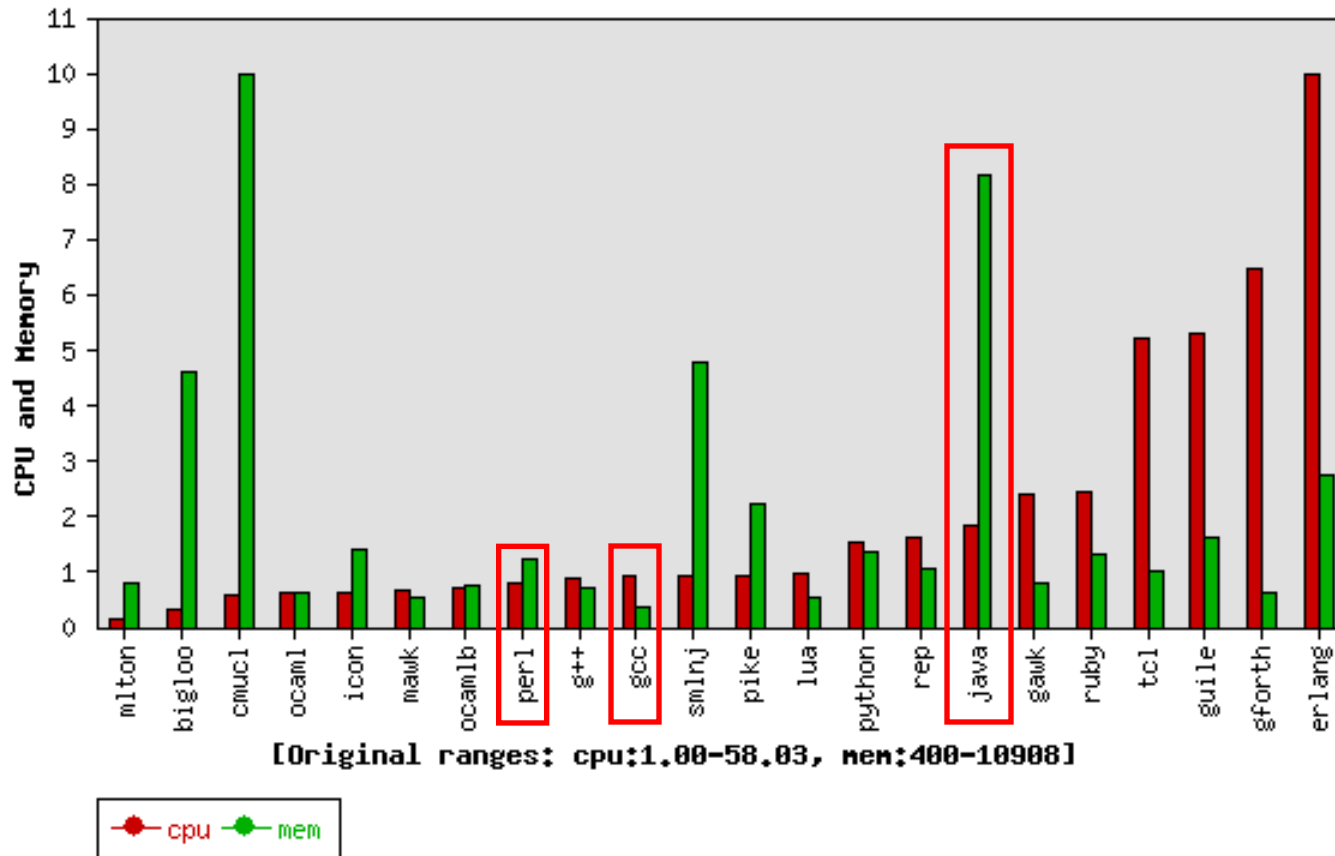
- Perl vs. Java
 - Both are highly portable
 - Java uses strict data typing, has more sophisticated data structure
 - Java is a true object-oriented language
 - Java is supported with Biojava initiative
 - Java recently introduced regular expression
 - Java has extensive standard APIs to facilitate development
 - Perl code is more concise – suitable for fast prototyping

The Great Computer Language Shootout

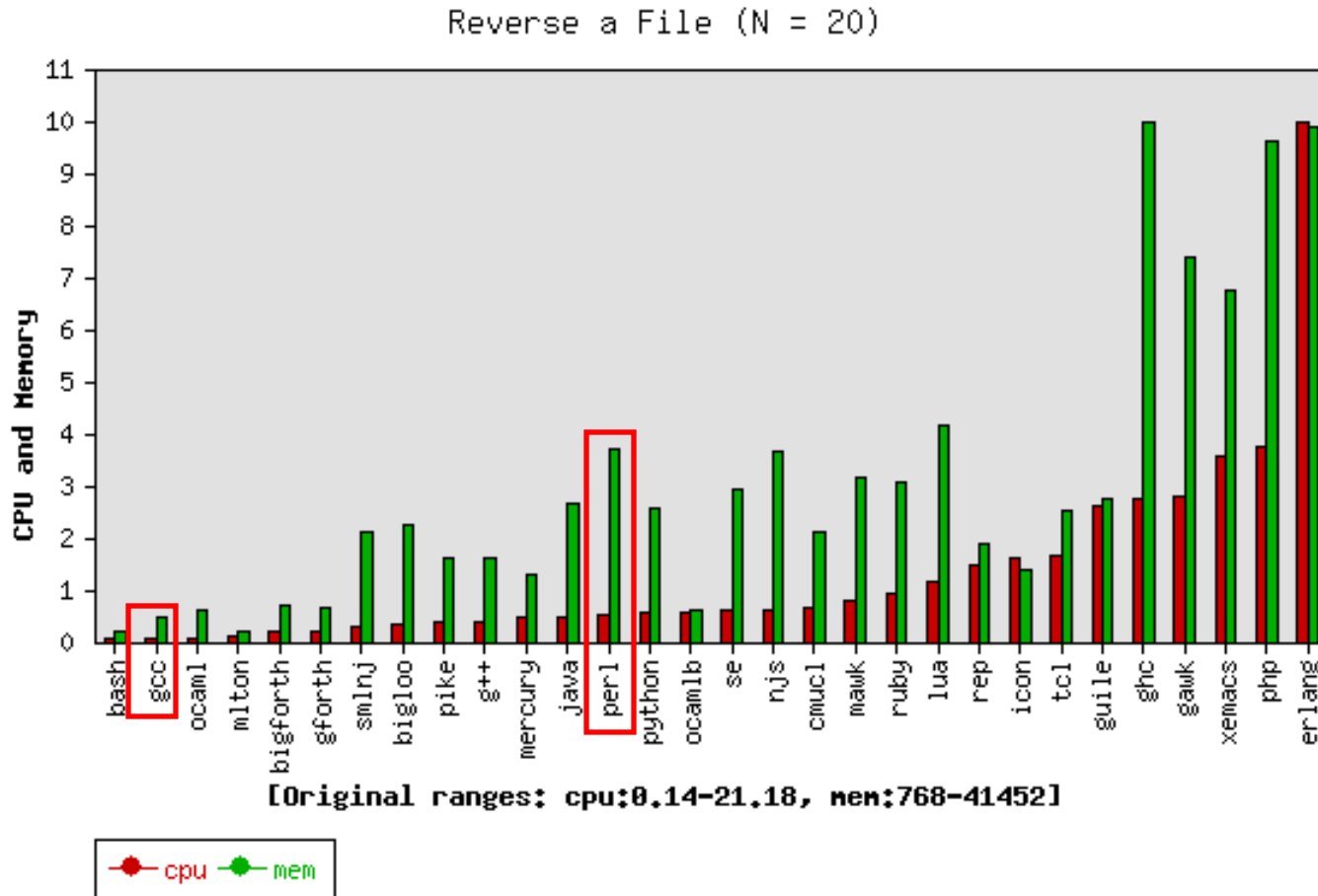
- A benchmark comparison of a number of programming languages (done in 2001)
- 30 Language Implementations, 25 Benchmark Tests, 750 Total Possible Programs, 632 Written
- Authour: Doug Bagley
- URL: <http://www.bagley.org/~doug/shootout/>
- Give an idea of how Perl measures up to other languages in different tasks

Shootout: REGEX

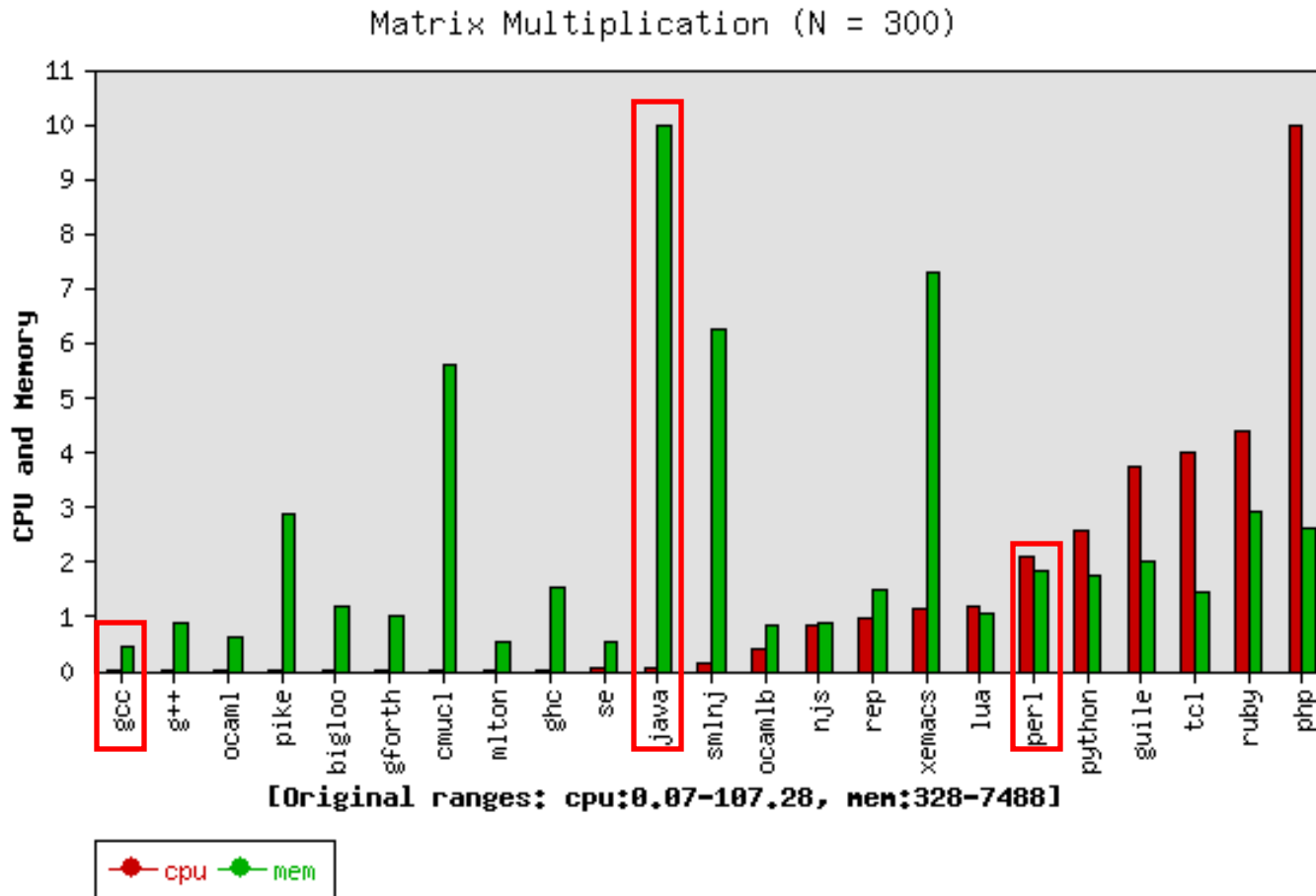
Regular Expression Matching (N = 9000)



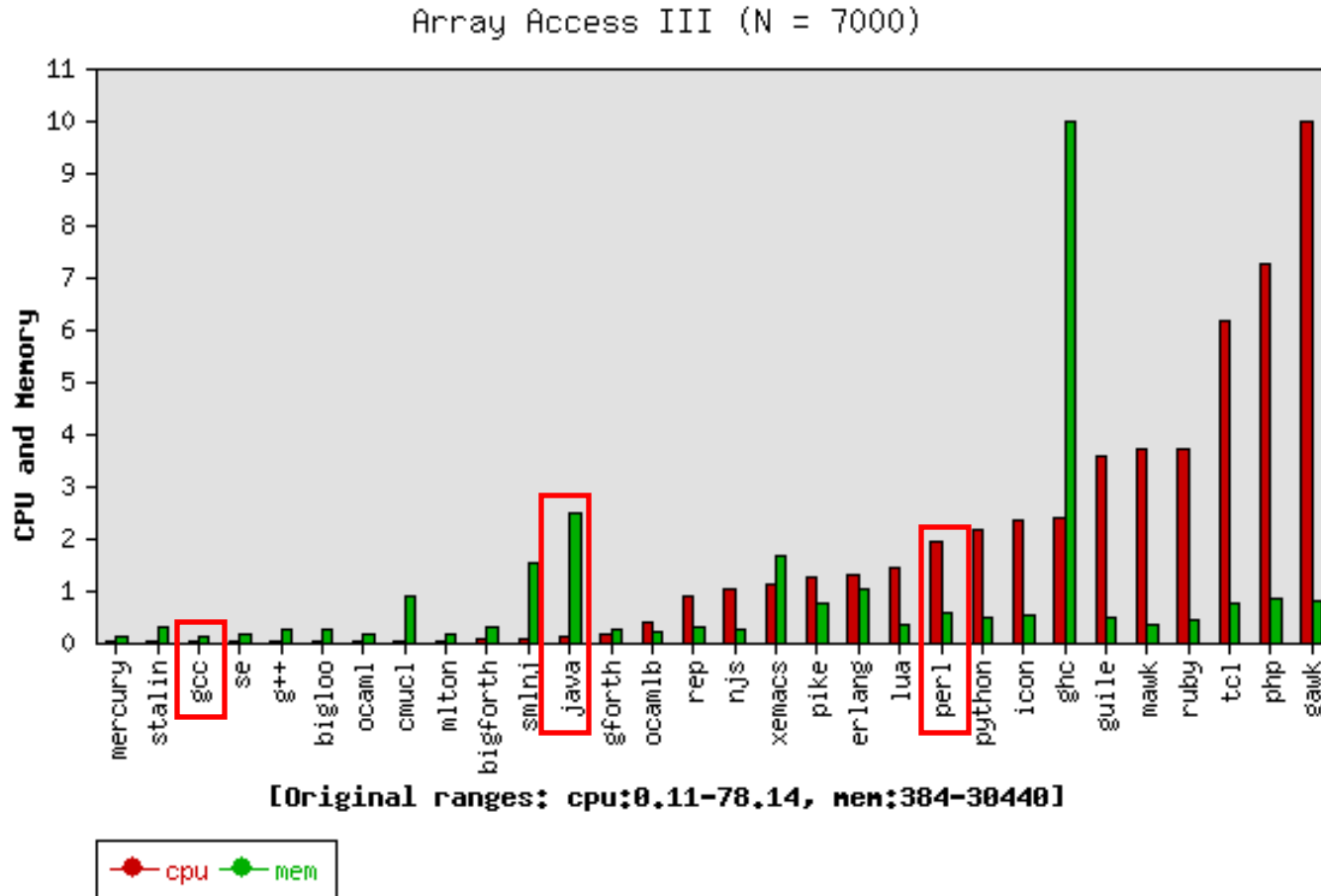
Shootout: File manipulation



Shootout: Matrix Multiplication

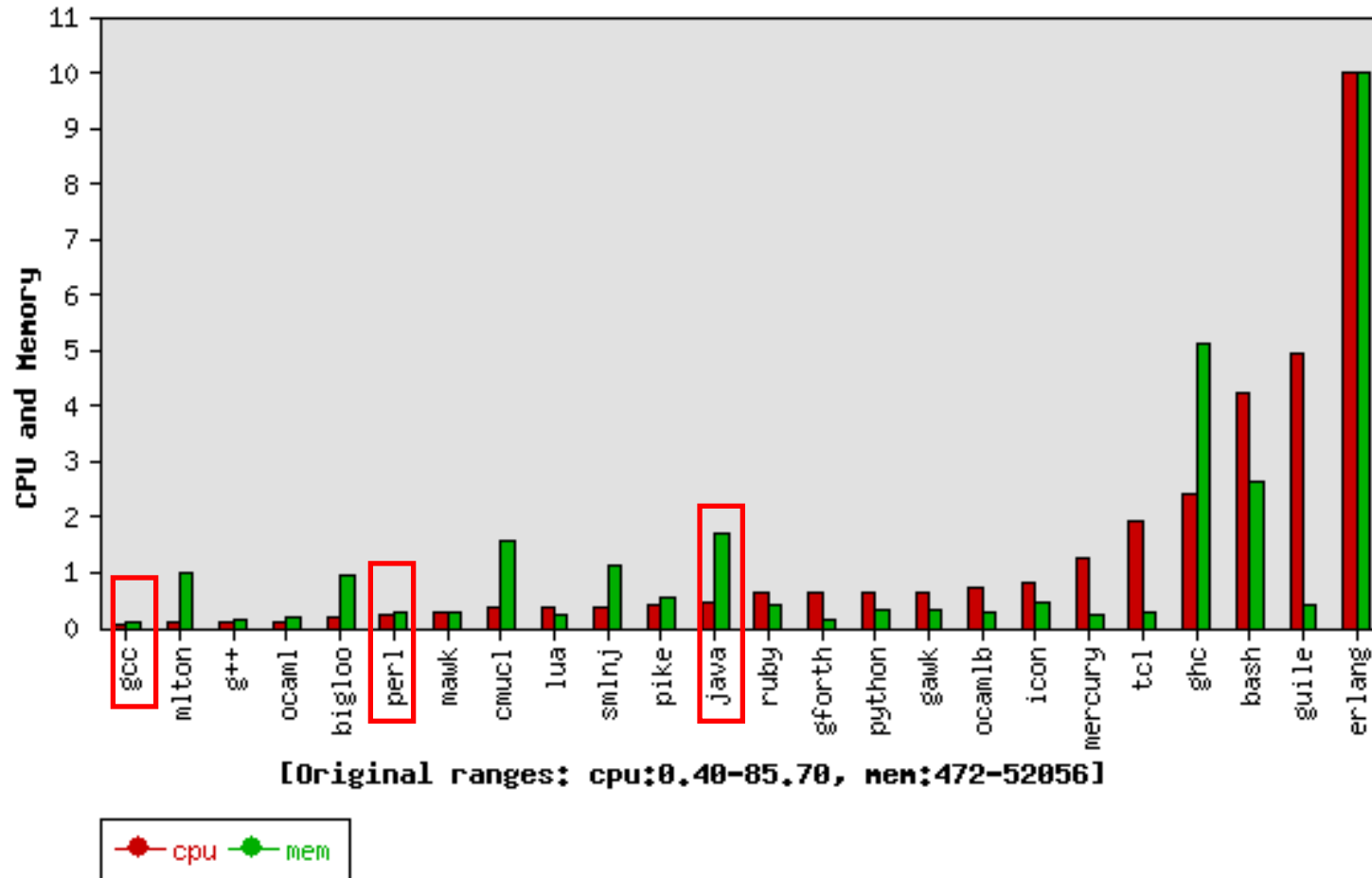


Shootout: Array Access



Shootout: Word Count

Word Frequency (N = 20)



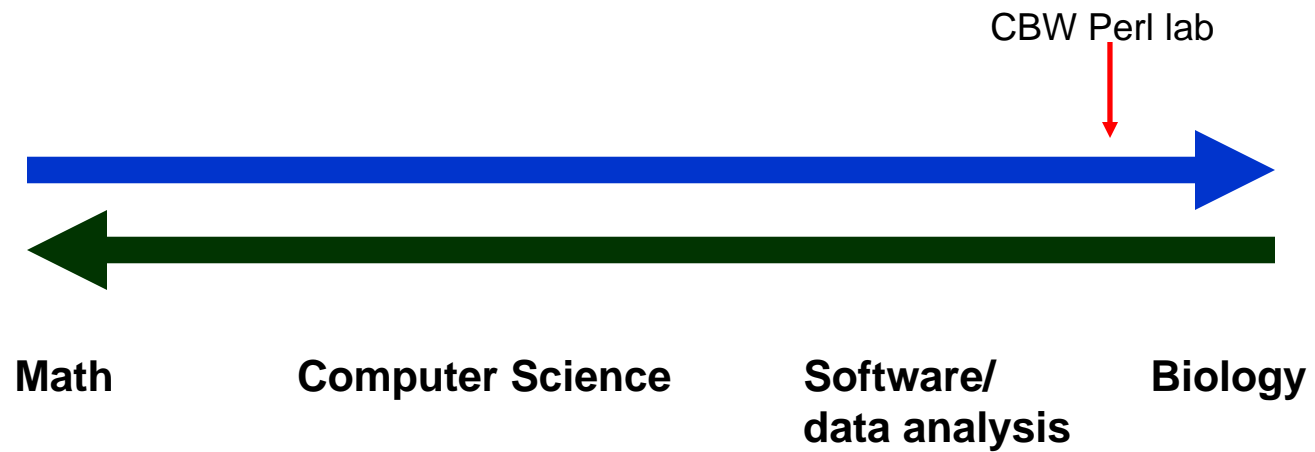
Perl vs. the world –bottom line

- Choose a language based on your needs:
 - Perl is NOT suitable for:
 - Applications requiring significant computation (number crunching)
 - Applications requiring sophisticated data structures that use large amounts of memory
 - Perl is suitable for:
 - Quick and dirty solutions (prototyping)
 - **Text processing**
 - Certain web applications and services (CGI based)
 - If you don't know C
 - Almost anything if performance is not an issue

Some Common Uses of Perl

- CGI.pm
 - Module for Common Gateway Interface by Lincoln Stein
- DBI.pm
 - Database Interface – allows communication between all major RDBMS systems (Oracle, MySQL, etc.)
- Net::FTP
 - Allows for automated scripting of data downloads
- REGEX
 - Complete set of tools for pattern matching text, for example:
/ ^ATG/ => begins with ATG

Bioinformatics Spectrum



Perl in Bioinformatics

- “How Perl saved the Human Genome Project”
 - Lincoln Stein (1996) www.perl.org
 - Perl allowed various genome centers to effectively communicate their data with each other
 - Introduces a project to produce modules to process all known forms of biological data

Bioinformatics cont'd

- The ***Bioperl*** project – www.bioperl.org
 - Comprehensive, well documented set of Perl modules
 - Last stable release 1.4.0
 - Open Source (Artistic License) project that has recruited developers from all over the world
 - Modules available for alignments (call BLAST, Clustal), sequence retrieval, annotations, sequence manipulation, gene prediction output, sequence databasing etc...
 - Stajich et al., The Bioperl toolkit: Perl modules for the life sciences. *Genome Res.* 2002 Oct;**12(10)**:1611-8. PMID: 12368254
 - Use with caution: things change fast

Bioperl code example

- Retrieve a FASTA sequence from a remote sequence database by accession #
- In 4 lines of code:

```
$refseq = new Bio::DB::RefSeq();  
  
$protein = $refseq->get_Seq_by_acc('NP_005329');  
  
$out = Bio::SeqIO->new('-file' => ">data/NP_005329.fa");  
  
$out ->write_seq($protein);
```

Bioinformatics cont'd

- The ***Ensembl*** project - www.ensembl.org
 - A software system that develops and maintains automatic annotations on eukaryotic genomes
 - Written entirely in Perl
 - Built on top of Bioperl
 - Is a major entry point into finding information about the human and other genomes
 - Hubbard et al. The Ensembl genome database project. *Nucleic Acids Res.* 2002 Jan 1;30(1):38-41. PMID: 11752248

Bioinformatics cont'd

- Bioinformatics in your labs:
 - Scripting – automation of repetitive tasks
 - Wrapping – accessing others programs (e.g. BLAST) through Perl
 - Web CGI'ing – Interactive WWW pages (user interface)

Running Perl

- Perl programs can be run in 2 ways
 - 1) invoking the perl interpreter explicitly
 - `unix_prompt> perl your_program`
 - 2) placing **'#!/path_to_perl_interpreter'** in the very first line of your UNIX program
 - Usually
 - `#!/usr/bin/perl`
 - `#!/usr/local/bin/perl`
- Don't forget to make your program executable!
 - `unix_prompt> chmod a+rx your_program`
 - `unix_prompt> chmod 755 your_program`

Perl Syntax

- Perl statements end with a semicolon ‘;’
- ‘#’ - means comment
 - The Perl interpreter will ignore anything after a # in a line (e.g. # this is a comment)
 - Comments are free – use ‘em!
 - Helps you and others understand your code
 - Critical in understanding cryptic Perl code
- Variables are preceded with \$, @, or % (e.g. \$sequence, @sequences)

A wee bit of code

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

```
# proudly exclaim our motto  
print "BKA!\n";
```

A Biological Example

Find the number of proteins in the yeast genome that contain a peptide cleavage site defined by:

[E|D]XXXXCS

- Search SGD (PatMatch)
- Download yeast.aa
- Write a small script

Declare to the operating system that this is a perl script

Use Bioperl **Modules**

Variable: holds the number of proteins containing the cleavage site

Display the result on screen

A Biological Example

```
#!/usr/bin/perl -w

use Bio::SeqIO::fasta;
use Bio::Seq;

$io = new Bio::SeqIO::fasta(-file => "$ARGV[0]");
$count = 0;
while ($seq = $io->next_seq()) {
    if ($seq->seq() =~ m/[E|D]....CS/) {
        $count++;
    }
}

print $count . "\n";
```

Answer?

326/6298

Watch out...

- Global variables (gasp!)
 - Can be used heavily in Perl and is the default mode for a variable
 - Can easily overwrite the value of a global inside a subroutine unintentionally
- No formal declarations of variables necessary
 - allows for typos
 - Good practice to “use strict vars” – forces variable declaration
- No strict datatyping
 - allows numbers to be exchanged for words, etc...
 - remember the context-sensitive nature of Perl
 - e.g. “2+3” (treated as number); “2 and 3” (treated as text)

Summary

- Perl is flexible, easy to use and can be applied to most problems
- Open Source with a huge user community
- Specialises in text processing
- Interpreted language so its slow for high volume or algorithmically complex data processing
- Used extensively in bioinformatics

Lab Preview

- You will convince Perl to:
 - Retrieve sequences from RefSeq
 - Retrieve files from a remote ftp server
 - Parse a text file
 - Format a FASTA database for BLAST
 - Run a BLAST search
 - Process the results of a BLAST search
 - Use your program to carry out one instance of “comparative genomics”

About the lab

- Self-contained WWW tutorial
- All code is provided
- All code is commented
- Understanding the exercises will be a huge amount of help in the assignment
- Work at your own pace
- Ask questions
- Discuss with your group but hand in your own assignment
- Link: http://www.bioinformatics.ca/bio/perllab_2004/

Perl lab Quick Ref

bioinformatics.ca **CBW::Bioinformatics::Perl Lab**

Sunday, February 23, 2003

Perl lab
[Setup](#)
[Exercise 1](#)
[Exercise 2](#)
[Exercise 3](#)
[Exercise 4](#)
[Exercise 5](#)
[Exercise 6](#)
[Exercise 7](#)
[Exercise 8](#)
[Summary](#)
[Assignment](#)

Useful links
[Perl Quick Ref](#)
[Net::FTP docs](#)
[GetOpt::Long docs](#)
[Bioperl](#)

Quick Reference Guide for Perl Lab

STDOUT The standard output stream. Usually this means to your console (or to the screen).

@ARGV System variable that contains the command line arguments a user enters when invoking a perl program.

. String concatenation operator: sticks the strings on either side of '.' together

\n Hard return character.

\t Tab character.

!= Means not equal to.

use Tells Perl you want to use functions and objects found in a specific module.

die Exits the Perl program

|| Literally means 'OR'

\$^T Returns the number of seconds since the 'epoch': a standard UNIX time reference that is exactly midnight on Jan. 1, 1970.

-e Exists operator. Check to see if a file or directory exists.

-M Modification time operator. Returns the number of days since a file was last modified.

== String matching operator. Does the string on the left of =~ match the regular expression on the right of =~ ?

^ 'Begins with' in regular expressions.

-> Call an object's method (or function).

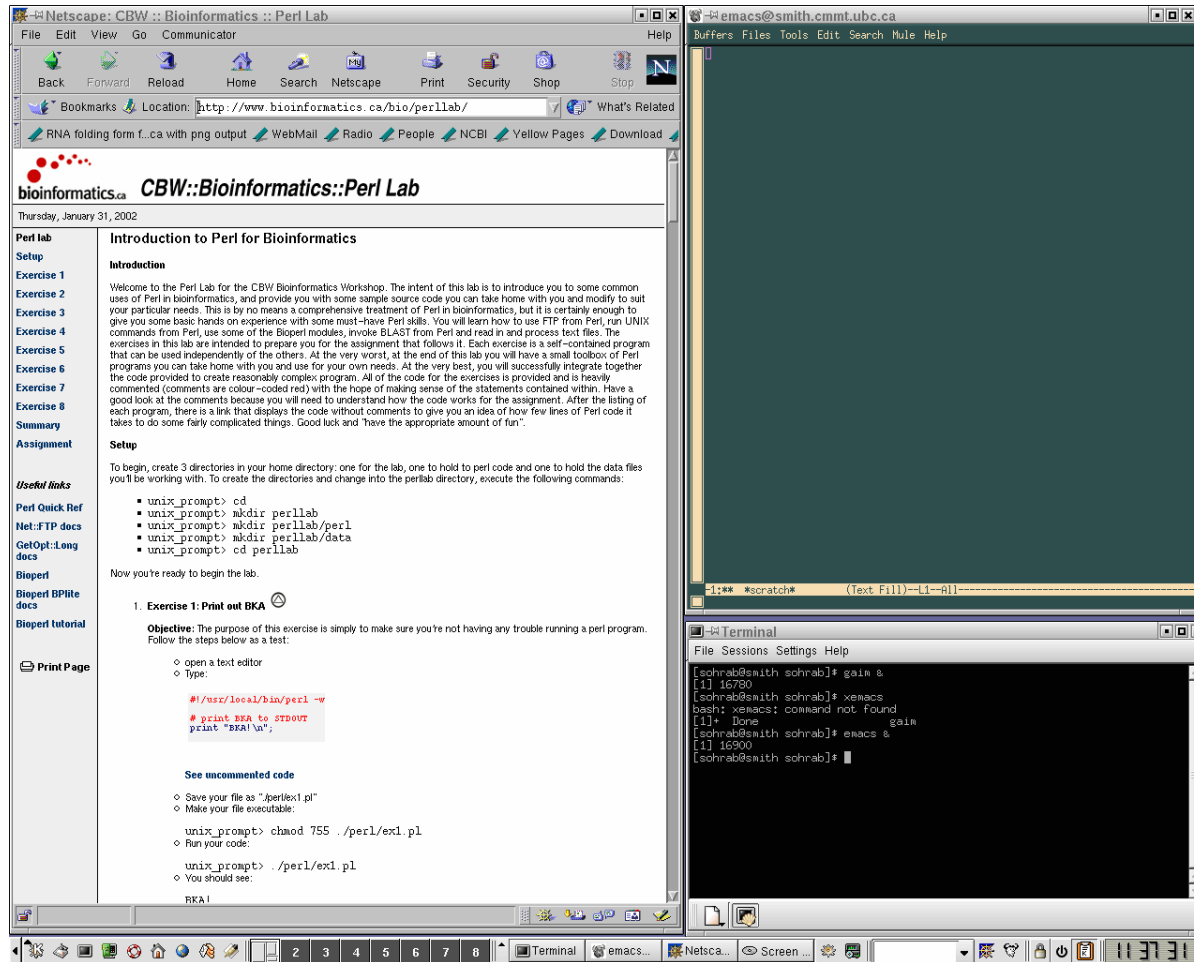
Perl Lab Frequently Asked Questions

- Question:**How do I make sure the right blast program (blastn, blastp, blastx, tblastn, tblastx) is being called given the query sequence and the database?
Answer:For the purposes of this assignment, you can assume the user will input the right program name given the molecule type of the query sequence and the molecule type of the database.
- Question:**How do I deal with the fact that when I gunzip my compressed database from NCBI, the filename no longer has the .Z extension?
Answer:In your command line argument containing the database, don't include the '.Z' extension. If you do this, be sure to add the '.Z' to the variable containing the remote file (for

Document: Done (1.127 secs)

Sample Desktop setup

browser



editor

console

URLs

- Perl
 - www.perl.com – O'Reilly
 - www.perl.org - Perl Mongers
 - www.cpan.org - CPAN – get modules for almost anything here
- Bioinformatics
 - www.bioperl.org
 - www.ensembl.org
- Perl people
 - www.wall.org/~larry - Larry Wall
 - stein.cshl.org/~lstein - Lincoln Stein
- Tutorials
 - <http://www.ugrad.cs.ubc.ca/~cs219/CourseNotes/Perl/intro.html>
 - www.bioperl.org/Core/POD/bptutorial.html
- Great Computer Language Shootout
 - www.bagley.org/~doug/shootout
- Open Source Licenses:
 - zooko.com/license_quick_ref.html –quick comparison

Thanks

- Sohrab Shah for the original slides, lab exercises
- Karsten Hokamp for inputs

wwhsiao@sfu.ca

Perllab FAQ

How does @ARGV work?

Unix/Linux Shell

- `>./program arg1 arg2 arg3 arg4`

Your Program

@ARGV



The rest of your program

Perllab FAQ

- What is “use *MODULE_NAME*” ?
 - *Tells Perl you want to use functions and objects in a specific module*
- What is “die (“Error message”)” ?
 - Tells Perl to exit the program name and print out an error message