The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts

Spring 2003

Introduction to Scripting Languages (with Perl)

David Stotts

Jan 27

1

Origin of Scripting Languages

- Scripting languages originated as *job control languages*
  - 1960s: IBM System 360 had the Job Control Language
  - *Scripts* used to control other programs
    - » Launch compilation, execution
    - » Check return codes
- Scripting languages got increasingly more powerful in the UNIX world
  - Shell programming, AWK, Tcl/Tk, Perl
  - *Scripts* used to combine *components*
    - » *Gluing* applications [Ousterhout, 97]

2

System Programming Languages

- System programming languages replaced assembly languages
  - Benefits:
    - » The compiler hides unnecessary details, so these languages have a higher level of abstraction, increasing productivity
    - » They are *strongly typed*, *i.e.* meaning of information is specified before its use, enabling substantial error checking at compile time
    - » They make programs more portable
  - SPLs and ALs are both intended to write application from scratch
  - SPLs try to minimize the loss in performance with respect to ALs
  - *E.g.* PL/1, Pascal, C, C++, Java

3

Higher-level Programming

- Scripting languages provide an even higher-level of abstraction
  - The main goal is programming productivity
    - » Performance is a secondary consideration
  - Modern SL provide primitive operations with greater functionality
- Scripting languages are usually interpreted
  - Interpretation increases speed of development
    - » Immediate feedback
  - Compilation to an intermediate format is common

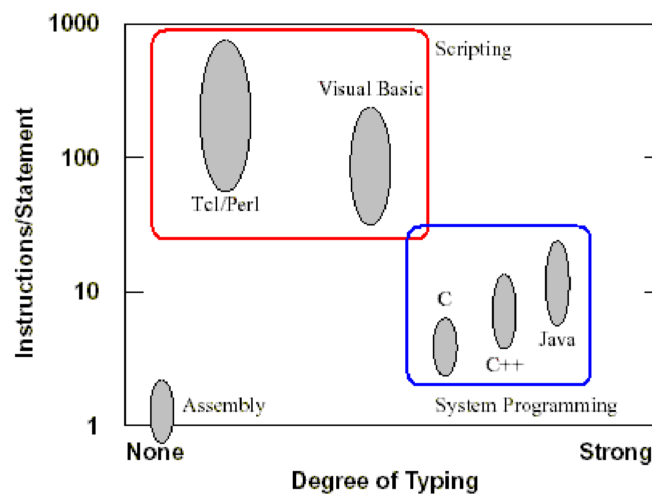
4

Higher-level Programming

- They are *weakly typed*
  - *I.e.* Meaning of information is inferred
  - ✗ Less error checking at compile-time
    - » Run-time error checking is less efficient, but possible
  - ✓ Weak typing increases speed of development
    - » More flexible interfacing
    - » Fewer lines of code
- They are not usually appropriate for
  - Efficient/low-level programming
  - Large programs

5

Typing and Productivity



[Ousterhout, 97]

6

Perl (Practical Extraction and Report Language)

- Larry Wall created Perl in the late 80s
  - *Wanted a notation that was more powerful than the Unix scripting facilities*
  - *Wanted linguistic "naturalness"… shortcuts, choices, defaults, flexibility*
- Perl is dense and rich
  - *"Swiss-army chainsaw"*
  - *"duct tape for the Web"*
  - *"there's more than one way to do it"*
  - *Experienced Perl programmers often reach for the manual when reading others' code*

7

Perl… goals

- Larry Wall on Perl
  - *…"I realized at that point that there was a huge ecological niche between the C language and Unix shells," says Wall. "C was good for manipulating complex things -- you can call it 'manipulexity.' And the shells were good at whipping up things -- what I call 'whipupitude.' But there was this big blank area where neither C nor shell were good, and that's where I aimed Perl."*
- Manipulexity *vs.* whipupitude

8

Brief Perl Timeline

- 1969 Unix created at Bell Labs
- 1977 awk is invented by Aho *et al.*
- 1978 "sh" shell is developed for Unix
- 1987 Perl is created by L. Wall
- 1995 Perl 5.001 released (up to about 5.8.0 now)

9

Perl Defined

- Original "man" page for Perl

Perl is (an) interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). It combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh, so people familiar with those languages should have little difficulty with it. (Language historians will also note some vestiges of csh, Pascal, and even BASIC|PLUS.) Expression syntax corresponds quite closely to C expression syntax. If you have a problem that would ordinarily use sed or awk or sh, but it exceeds their capabilities or must run a little faster, and you don't want to write the silly thing in C, then perl may be for you. There are also translators to turn your sed and awk scripts into perl scripts.

OK, enough hype.

10

What Perl Does Well

- String manipulation
- Text processing
- File handling
- Regular expressions and pattern matching
- Flexible arrays and hashes
- System interactions (directories, files, processes)
- CGI scripts for Web sites

Has objects, pointers, threads, etc. but these are not the original design goals

11

Perl Overview

- Perl is interpreted
  - *Actually, compiled to bytecode and the bytecode interpreted*
- Every statement ends in semicolon
- Comments begin with # and extend one line
- Perl syntax is considered convoluted by some, elegant and efficient by others

12

Built-in Data Types

- No type declarations
  - *Types are distinguished lexically (by first character)*
- Perl *does not* have integer, float, boolean, etc. *types* like other languages
  - *In Perl, these are values of type Scalar*
- Perl has 3 types:
  - *Scalar*
  - *Array*
  - *Hash (associative array)*

13

Built-in Data Types: Scalar

- Scalar
  - Integer, real, boolean, string values
  - Scalar variables begin with $
  - $a $A $var1 $fooDeeBar $_   *# case matters*
  - $a = 5 ;
  - $a = 3.5;
  - $a = "hi there" ;
  - $a += 2 ; *# what happens here?*

14

Context… conversions

- When a scalar is used the value is converted as appropriate for the context
  - $a = 5 ;
  - $a = $a . " is a good number" ; *# string concatenation*
  - print $a ;
  - $a = "3.7" ;
  - $b = $a + 43 ;
  - print "$b \n"; *# string in $a is treated as real number*
  - print '$b \n' ;

15

Built-in Data Types: Array

- Array
  - array variables begin with @
  - @a @var1 @fooDeeBar
  - @a = (9, 5, 7.1, "last elt") ;
  - $a[1] = 12 ;
  - print "first element is $a[0]\n";
  - print $#a ; *# tells largest used subscript*
  - @b = @a ;

16

Built-in Data Types: Hash

- Hash
  - hash variables begin with %
  - They are also distinguished by use of { } for subscripts
  - Subscript can be any scalar (usually a string)
  - %a = ("first", 43, "second", 26, "third", 17) ;
  - print $a{'second'} ;
  - $a{'seventeenth'} = 3.1415926 ;
  - $a{'first'} = 41 ;
  - $z = "third" ;
  - $a{$z} = "$z place" ; *# what does this do?*

17

Built-in Data Types: Hash

- Hash
  - Hashes are very useful for text processing
  - Build tables, lists, etc.
  - Built-in functions for getting list of all subscripts (called keys)
  - For example

```
%a = ( "first", 21, "seventeenth", 41, "2nd", 27, "1", "one" );
foreach ( keys(%a) ) {   # loads built-in var $_ with a key
     print "( $a{$_} ) : $_  \n" ;
}
```

18

Reading Assignment

- John K. Ousterhout, *Scripting: Higher-Level Programming for the 21st Century*, 1997
  - http://home.pacbell.net/ouster/scripting.html
- D. Stotts, *The PERL Scripting Language*, 2003
  - http://rockfish-cs.cs.unc.edu/COMP144/IEPerl.pdf

19