Overview

- Random C++
- Regular Expressions
- Search engine technology
Announcements

- Last week of classes
- Wrapping up C/C++
  - Familiar with language and development
  - Managing projects with IDEs and makefiles
- CGI
  - Programming over the web
- Concepts
  - Pointers and References
  - Memory management
  - Basic scripting with perl
  - Regular expressions…
Classes

- Remember that C++ is pass by value

- Now…
#include <iostream.h>
using namespace std;
class X {
    public:
        X() { cout << 1 << ' '; } 
        X( const X& ){ cout << 2 << ' '; } 
        ~X(){ cout << 3 << ' '; } 
        X& operator=( const X& ){ cout << 4 << ' '; }  
    
    X f( X x ){ return x; } 
    X& g( X& x ){ return x; } 
};

int main() {
    X a; 
    X b = a; 
    cout << endl; 
    a = b; 
    cout << endl; 
    a = f( b ); 
    cout << endl; 
    b = g( a ); 
    cout << endl; 
    return 0;  }

Can you predict the output here? Explain why?
Perl

- Practical programming language
- Aims to be helpful
- Very easy to work with files (mentioned last time) and find patterns (today’s stuff)
Variables are created and setup for you as it runs

- Scalars $something
- Lists @something
- Hash %something

Memory is allocated automatically (not necessarily most efficient)

Great for prototyping concepts…
Take a second to look this over

#!/usr/bin/perl
open ( TEST, "test.txt" ) || die "can not open file!\n";
$linecount = 0;
while ($line = <TEST>){
   $linecount++;
}
close( MYFILE );
print "number of lines in the file: $linecount\n";
Some built in functions..

- `chomp $var`
- `chomp @list`
removes any line-ending characters

- `chop $var`
- `chop @list`
removes last character

- `chr number`
returns the character represented by the ASCII value number

- `eof filehandle`
returns true if next read on filehandle will return end-of-file

- `exists $hash{$key}`
returns true if specified hash key exists, even if its value is undefined

- `exit`
exits the perl process immediately
Different subs

- Perl will make a best effort to both figure out what the variables mean and which functions you are trying to call

- `sub name {    }
  is a subroutine which can be called using name(); or &name();`
Sample #1

#!/c:/perl/bin

($first,$last) = &getname();

print "First is $first";

#return the full name as a string
sub getname(){
    return "shlomo hershkop";
}

#return name split
sub getname(){
    return ("shlomo","hershkop");
}
Perl copy function…

#!/usr/bin/perl
open( MYFILE1, "a.dat" ) || warn "file 1 not found!";

open( MYFILE2, ">b.dat" ) || die "file 2 problems!";

while ( <MYFILE1> ) { print MYFILE2 "$_\n" }

close( MYFILE2 );
Not only read line

- Can also read individual bytes from files (useful in low level graphic manipulation)

- `getc filehandle`
  reads next byte from filehandle
Pragmas

- Perl allows you to control the interpreter
- Pragmas are compiler hints to allow you to operate in some special mode

- Two examples:
  - use warning
    - This will give you ideas of what perl thinks it is doing
  - use strict
    - This will be very strict about variables, so you will need to declare each variable (my, our) before being able to use it in your program (can declare it on same line you are using it)
Strict mode

- This isn’t about the final exam!
- Tells perl to only allow variable you explicitly create in your programs
  - Prevents typos
  - Easier to maintain
  - Less work for interpreter
  - Will clearly state what it thinks you need to be doing to get things correct
Use strict

- Goes at top of your program

- Example for loop:

```perl
for( $i =0; $i < 100; $i++)  
  won’t work if first use of $i
for (my $i; $i < 100; $i++)  
  will work for strict, but scoped locally
```
Good advice

- Learn to read errors and warnings
  - Will tell you what the problem is and what line it thinks it's on
  - Use perldoc -f

- Do not ignore or twiddle...see why an error is occurring
Ok lets switch gears

- You now have enough knowledge to code some simple but powerful programs

- But now the questions becomes:

- How to program patterns…
  - “Extract all credit card info from the computer”..
Patterns

- Many times when you process text, you need to find patterns
  - Email addresses
  - Phone numbers
  - Credit cards

- How would you look for all phone numbers in a text file using java?
Regular expressions

- Regular expressions is an elegant way of expressing patterns

- Use of simple building blocks to express even the most complex patterns
Simplest pattern

- A string is the most simple pattern

- “computer”

- Is the pattern c followed by o followed by m etc

- Will not match to “cosputer”
Matching patterns

- Matching function
  `m/  /;`
- Pattern goes between slashes, m is optional
- Returns true if match occurs, false otherwise
- When you want to match against text use the `=~` operator

```perl
$sometext =~ m/computer/;
```

will look for any occurrence of the word computer in the scalar string

```perl
if ($name =~ m/shlomo/) { print “shlomo here”;}
```
Basic

What will this do?

```perl
#!/c:\perl\bin
$name = "shlomo hershkop";

if($name =~ /lom/){
    print "have found match\n";
} else{
    print "no match found\n";
}
```
modifiers

- The matching operator can be modified by following the last slash with specific characters
- i = matches case insensitive
Patterns I

- You can force where on the line the pattern is match by

- ^ = start of the line
- $ = end of the line

```perl
$line =~ /^credit/  # line begins with word credit
$line =~ /bye$/     # line ends with word bye
```
Flip patterns

- Can flip the match by saying !~

```
$line !~ /^great/i
```

will match any line not starting with the word great (case insensitive)
substitute

- Related function
  - Instead of matching operation, can use substitute
  - `s/ / /`
    - first space is pattern to find
    - second space is pattern to replace it with

- Returns the number of times substituted
  - `$n = ($something =~ s/credit/XXXXX/);`
Example

```perl
#!/usr/bin/perl
$s = "hello world";
print "$s=[
";
print "$s ]\n";
if ($s =~ m/x/)
    { print "there's an x in ",$s,"\n" }
else
    { print "there isn't\n" }

if ($s =~ m/L/i)
    { print "there's an l in ",$s,"\n" }
else
    { print "there isn't\n" }
```

输出：
```
$s=[hello world]
there isn't
there's an l in hello world
```

猜测输出…
Example 2

```perl
#!/usr/bin/perl
$s = "hello world";

$t = ($s =~ s/l/x/g);

print '$t=[', $t, ']
print '$s=[', $s, ']
```
Next level up

- Complex regular expressions use *metacharacters* to describe various options in building a pattern.

- \ 
  - Escape character that is backslash followed by something has special meaning, like \n
- . 
  - Match any single character
  - /compu.er/
    - can match computer and compuser

- * 
  - Match zero or more times whatever is before the star
  - /comp*uter/
    - will match computer and compppppputer and comuter
More modifiers

- *
  - Will match 0 or more times..

- ?
  - Will match 0 or 1 times

- +
  - Will match 1 or more times

/ab+a/

will match aba and abba abbbbbba but not aa
Escape shortcuts

\w  Match "word" character (alphanumeric plus "]")

\W  Match non-word character

\s  Match whitespace character

\S  Match non-whitespace character

\d  Match digit character

\D  Match non-digit character
Other escape codes

\t Match tab
\n Match newline
\r Match return
\f Match formfeed
\a Match alarm (bell, beep, etc)
\e Match escape
Regular expression attributes

Attributes follow the reg exp (as we saw with case insensitive)

- **g** = match globally (all instances)
- **i** = do case insensitive matching
- **e** = evaluate right side as an expression
- **s** = let . match newlines
- **m** = $ and ^ can refer to inside newlines
- **c** = compliment
usages

1) if ( $line =~ /^\s.*\S$/ ) {....}

2) if (not $line =~ /cs3157/ ) {…}
   if( $line !~ /cs3157/ ) {....}

3) while ( $line =~ /^\w \w$/ }
groups

To allow groups of alternative choices use pipe

```perl
if($string =~ /(A|E|I|O|U|Y)/i)
  { print "String contains a vowel!\n"; }

if($string =~ /(Clinton|Bush)/)
  { print "President sir!\n"; }
```
Character choices

we can also specify character ranges by using the square brackets:

```perl
if( $string =~ /[AEIOUY]/i ) {
    print "String contains a vowel!\n";
}
```

Can also specify ranges
```perl
if( $string =~ /[^a-e]/i ) {
    something
}
```
Groups II

- Perl has shortcuts to allow us to reference for selection and substitution
- Each group can be referred to by scalar $1, $2, $3 ....

Example

```perl
元line = "From s@aol.com Wed Jun 3 12:12:12 2005"
if($line =~ /^From (.*) (…) (…) (.*)$/)
```
quantifiers

- \( ba^*b \)
- \( ba\{3,5\}b \) between 3 and 5 a’s
- \( ba\{2\}b \) exactly 2 a’s
- \( /(ab)\{4,\}/ \) 4 or more ab’s
Shortcut 1

- We can say
  - [abcdefgfh]
  - [a-h]
  - [a-h] \{1,4\}
1. How would we look for a phone number using regular expressions??

2. What about a street address?
Quick question

- How to indicate the period since period matches any character?
open MAIL, "mail.txt" or die "cant open file\n";

while(<MAIL>) {
    print if m/^From: /;
}
open MAIL, "Mail.txt" or die "can't open mail file\n";

while (<MAIL>) {
    if (/^([^:]+): (?(.+))$/ ) {

        print "Header $1 has val $2\n";
    }
}
Other shortcuts

$name = "advanced programming class"

if($name =~ /programming/){
    print $` ;
    print $` ;
    print $` ;
    print $` ;
}

What is scope?
scope

- Default scope is main
- $name can also be referred to as $main::name

- package NAMESPACE
  - Within any block of code, can declare that the rest of the code will belong to a specific namespace
Scope II

- my
  declares the variable and value local to the current scope
- our
  confines the name to local scope
- local
  confines the value to local scope
- Remember to place more than one variable in parenthesis!!

- my $time, $out
- #here only time is a local variables

  my ($time,$out)
  #correct way to do it.
Security

- Should use pattern matches as a security check on input

  Example:

  ```
  unless ( $year =~ /^\d\d$/) {
    die ("problem with year input!");
  }
  ```
Argument passing

- Can get command line args:
  - @ARGV

- How to get the length?
- How to get individual items?

- Lets write a short program to print them all out
Unix File system

- Everything is a file
- Root (top) level file is slash /
- Everything is under root
- General configuration is usually in /etc/
- The login information is in /etc/passwd

- Actual passwords:
  - Used to be plaintext in /etc/passwd
  - Later scrambled ....
  - Later put in private shadow file
permissions

- Different levels of permissions allow a wide variety of control over underlying operating system

- User/groups/others
- Read/write/execute

- `ls -la [filename]`
- `chmod [ugoa][+-][rwx]`
Code examples

- We want to process the `/etc/password` file
  - Any idea what would be a good application here?
- Looks like:
  - `pcap:x:77:77:ARPWATCH User:/var/arpwatch:/sbin/nologin`
  - `ident:x:98:98:pident user:/bin/false`
  - `nobody:x:99:99:Nobody:/sbin/nologin`
  - `xfs:x:405:405:X Font Server:/etc/X11/fs:/bin/false`
  - `mysql:x:6730:1101:mysql server:/var/lib/mysql:/bin/bash`
sub read_passwd {
  my %users;
  my @fields = qw/name pword uid gid fullname home shell/;

  while(<STDIN>) {
    chomp;
    my %rec;

    @rec ={@fields} = split(/:/);

    $users{$rec{name}} = \%rec;
  }
  return \%users;
}

my $users = read_passwd();

my @names;

foreach (keys %{$users}) {
    next unless $users->{$_}{fullname};

    my ($fname, $lname) = split (/\s+/,
        $users->{$_}{fullname},2);

    push @names, "$fname $lname";
}

print map { "$_\n" } sort @names;
Random helpful stuff

- $| = 1$
  will turn off output buffering great when working with cgi (later today)

- In perl, can call external commands i.e.
  we can execute command line arguments
  1. Backticks (``)
  2. System
  3. exec
Perl + cgi

- Remember:
  - Perl is only a tool here

- STDIN
  - Contents passed to perl script

- STDOUT
  - Will need HTTP headers before printing

- STDERR
  - Depends on server, sometimes just error logs, sometimes error reports on client

- Don’t forget to setup the webserver.....abyss in this case...
%ENV

- This is your best friend in PERL CGI
- Way of getting information **from** the client
- Create content is way to pass back information **to** the client
Remember

- Need to set permissions:
  - `chmod 0755 ???_.cgi`
  - `-rwxr-xr-x`

- Need to place script in correct place
  - Usually `cgi-bin/` directory

- Naming
  - Usually need to end in `.cgi`
Problem

- How can we print out all the environment variables?
Example

#!/usr/local/bin/perl

use strict;

my $vars
print "Content-type: text/html\n\n";

foreach $vars (sort keys %ENV){
    print "<P><B>$vars</B><BR>
    print "$ENV{$vars};
    }

Code problem….

- Lets talk about a common programming problem

- Task: Word statistics:
  - Count the number of unique words in a given file.
  - Top 5 occurring words in the file
  - Count the average length of words in the file.

- Many different approaches, lets do one of them now…. 
hashes

- A hash function is a function that converts an input from a (typically) large domain into an output in a (typically) smaller range.

- Example:
  - Map each name in the class to a somewhat unique number.

- Collision = when different keys map to the same output.
Use of hashes

- Hash tables
  - Data structure
  - Unordered list, fast lookup
- Cryptography
- Data processing
Short check

- So you have enough knowledge to process a file and find all emails

- Can you take a second to write a short pattern on what an email looks like?
Useful command

- Split

split /PATTERN/,EXPR,LIMIT
split /PATTERN/,EXPR
split /PATTERN/

split  Splits a string into a list of strings and returns that list. By default, empty leading fields are preserved, and empty trailing ones are deleted. ..............
Perl References

- there are lots and lots of advanced and funky things you can do in perl; this is just a start!

here’s a quick start reference:
- http://www.comp.leeds.ac.uk/Perl/
- http://www.perl.com

function reference list is here:
Sketch

- Hash table idea

- If we don’t care about order of insert can get truly fast organization of list items
Search engine technology

- Generally search engines work in the following way:
  - collect webpages
  - index information
- wait for user to search
  - Return relevant results
So how can we quickly find results?

That is match the terms to the websites pages?
Vector Model

- Each document is a vector in an n-dimensional vector space of search terms.
- Take query and find closest points.
- Sparse (very).
- If one word tokens, order will be ignored.
- Need to generate a master word list
  - Can run through a bunch of examples for this
- can strip out stop words
  - Std lists exist
- can also calculate related words i.e. runs and run worry and worrying through “stemming”
Example: master word list

- cat
- dog
- fine
- good
- got
- hat
- make
- pet

# A cat is a fine pet
$vec = [ 1, 0, 1, 0, 0, 0, 1 ];
many ways of getting similarity between two documents

- Cosine distance
General issues

- Better parsing
- Non-English Collections
  - stemming
  - stop words
- Similarity Search
  - can combine a few docs to find similarity
- Term Weighting
- Incorporating Metadata
- Exact Phrase Matching
hash table of search words

inverted index table

- Table of words and then list of documents which they appear in..
- Can add freq and locations