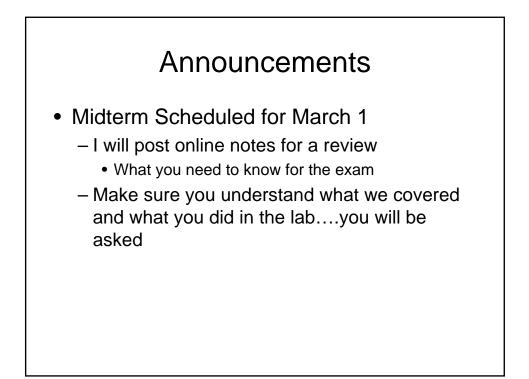


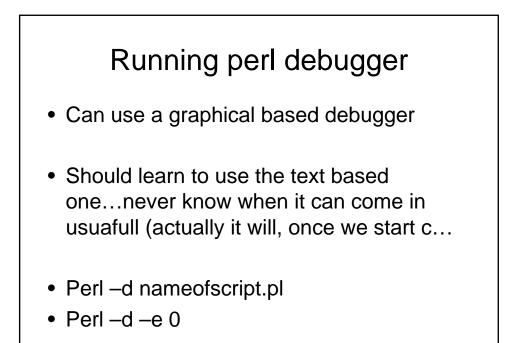
#### Announcements

- HW1 is going to be due soon
- Office hours + email + IM
- Reading: please wrap up the perl reading and start on C (make sure you have a book by the end of the week and read (see class page for reading list)



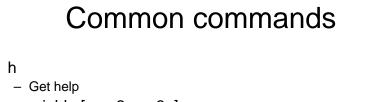
## Debugging process

- This is a general programming idea:
- We have some code instructions, would like to examine them as they run:
- 1. Output test cases for each line (hope it doesn't crash)
- 2. Run it within another program allowing us to fine tune control of running process and interaction with the running environment

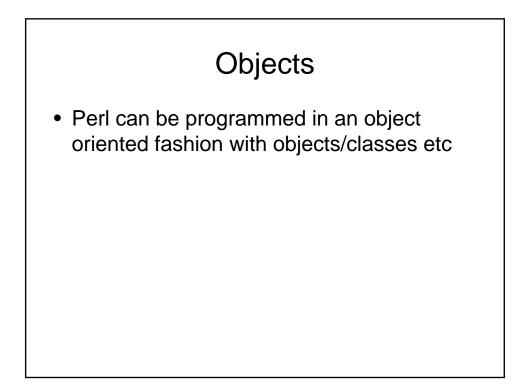


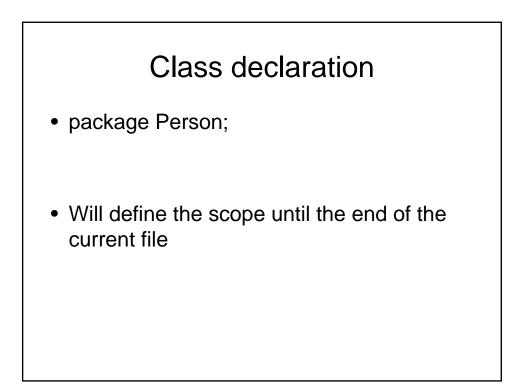
### What you see

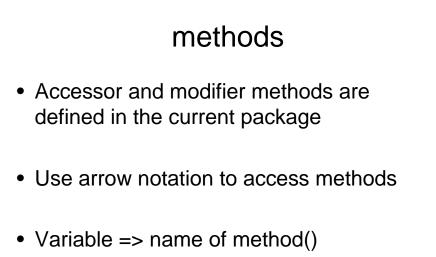
- Current scope (main::)
- What line it will execute next
- No need for comma, 'enter' key is a signal
- · Can do many things
  - Evaluate expressions
  - Check variables
  - Step through code



- x variable [, var2, var3..]
  - Examine a variable (or set)
- p
  - To print something
- V ???
  - Examine all variables in scope/package ?? (example main)
- s
  - Step through next instruction (including sub)
- n
  - Jump over subs



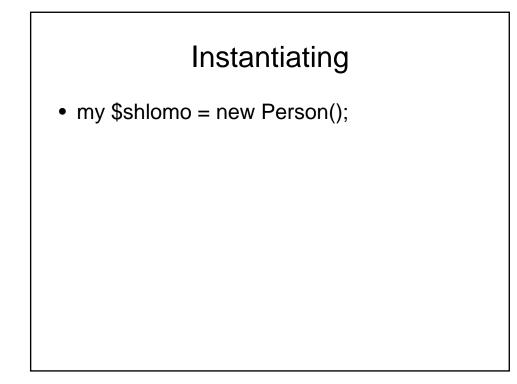




```
sub print {
  my ($self) = @_;
  #print info
  print $self->firstName . " ". $self-
  >lastName;
  }
```

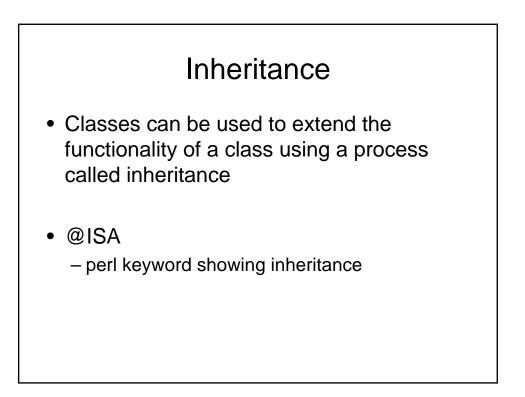
#### constructor

```
sub new {
  my $self = {
  _firstName => undef;
  _lastName => undef;
};
bless $self, 'Person';
return $self;
}
```



#### accessing

```
sub firstName {
  my ( $self, $firstName) = @_;
  $self->{_firstName} = $firstName if
  defined ($firstName);
  return $self->{_firstName};
 }
```



# Text handling

- One of the exciting developments in the last decade of computer science is data processing/mining/learning
- Many other area in and out of CS need data to be analyzed or presented in some (controlled but arbitrary fashion)

### Handling data

- Using chiseled stone
- By hand (literally copy paste)
- Early mechanics (typwriters)
- Take 3157 🙂

# Outputting text

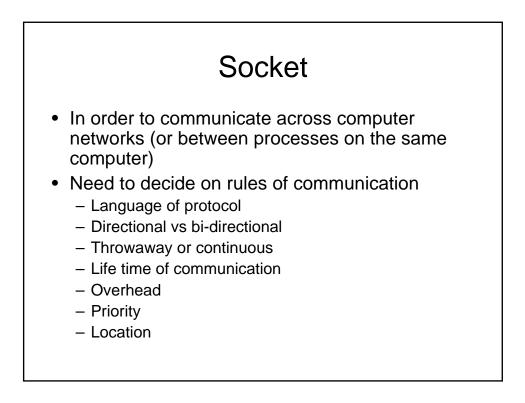
- Many times will have multiple fields per line
- Arbitrary delimiters:
  - Comma
  - Tabs
  - Pipe |
- Make sure whatever you choose
  - Is either not/can't be present in the data
  - What if it is? How to represent these delimiters ??



- Memory vs disk based handling
- Brute force
- Divide and conquer
- Regexp is your friend

# Ahead!

• Because CGI/Internet involves network based thinking, I will illustrate a quick example now.



### **IO:Socket client**

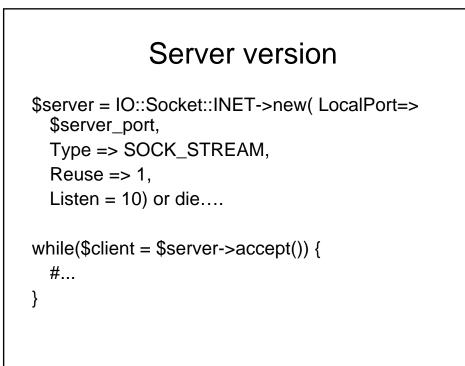
```
Use IO::Socket::INET;
```

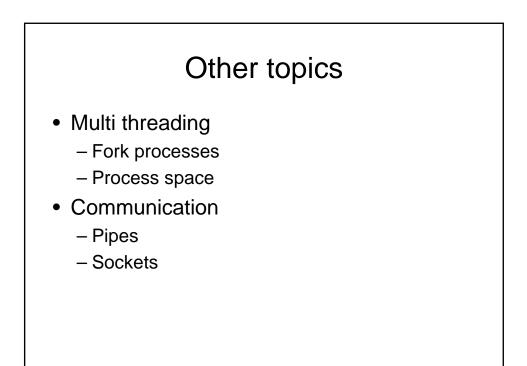
```
$socket = IO::Socket::INET->new(
    PeerAddr => $remote_host,
    PeerAddr => $remote_port,
    PeerAddr => "tcp",
    PeerAddr => SOCK_STREAM) or die...
```

#writing out
print \$socket "hello World";

\$answer = <socket>;

close(\$socket);





## **Testing environment**

- Many times need to test code
- Large projects
- Bugs cost time and money
- Bugs hurt morale
- Human are programmers...humans make mistakes

# Automated testing

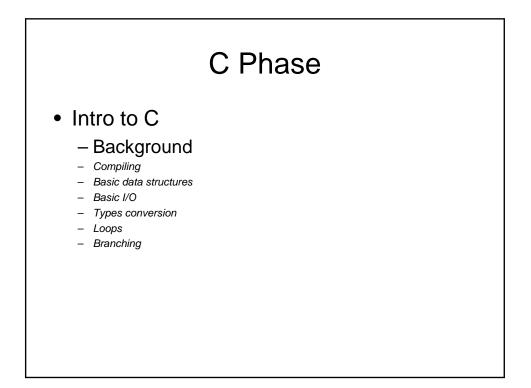
- Humans hate testing...
- Fast verification that new feature has not broken code
- Verify all code on a regular basis
- No grumble if test to rerun test ©

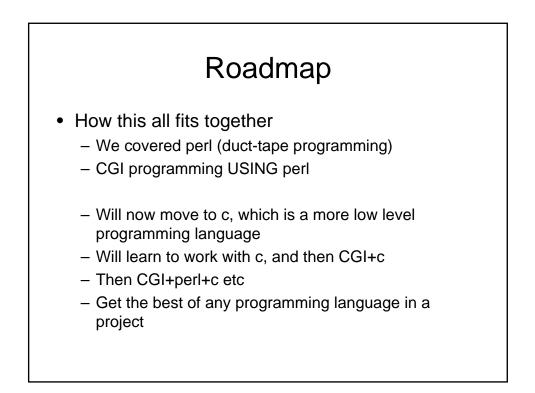
## packages

- There are packages out there (Test::Simple and Test::Harness) to automatically run tests
- Verify what happens on good/bad input
- Verify variables/method behavior
- Usually .t files have tests

# Shift gears

• Will now start c component of the course





# Why Learn C ?

- C provides stronger control of low-level mechanisms such as memory allocation, specific memory locations
- C performance is usually better than Java and usually more predictable (very task dependant)

# Why Learn c continued

- Java hides many details needed for writing code, but in C you need to be careful because:
  - memory management responsibility left to you
  - explicit initialization and error detection left to you
  - generally, more lines of (your) code for the same functionality
  - more room for you to make mistakes
- most older code is written in C, will need it if upgrading or interfacing

# Background

С

- Dennis Ritchie in late 1960s and early 1970s
- systems programming language
- make OS portable across hardware platforms
- not necessarily for real applications could be written in Fortran or PL/I



C++

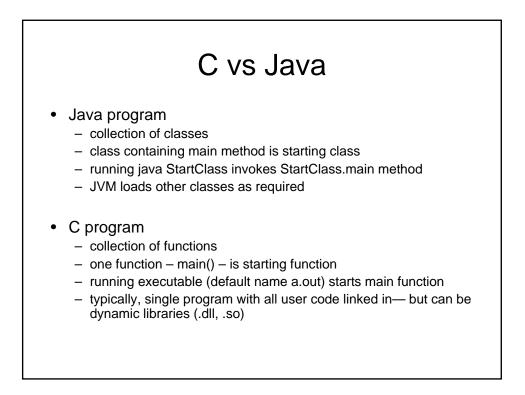
- Bjarne Stroustrup (Bell Labs), 1980s
- object-oriented features

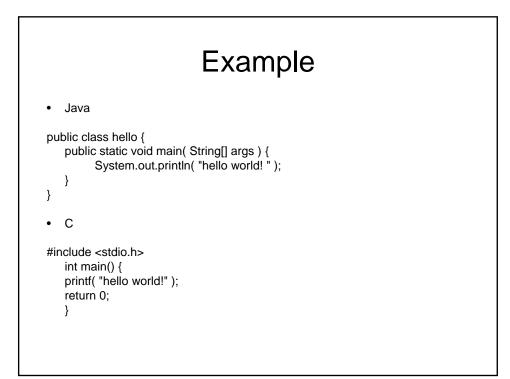
Java

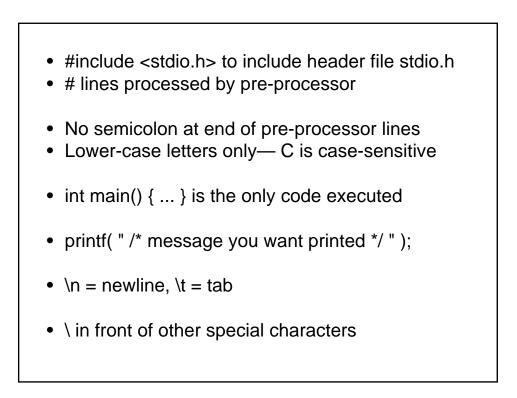
- James Gosling in 1990s, originally for embedded systems
- object-oriented, like C++
- ideas and some syntax from C

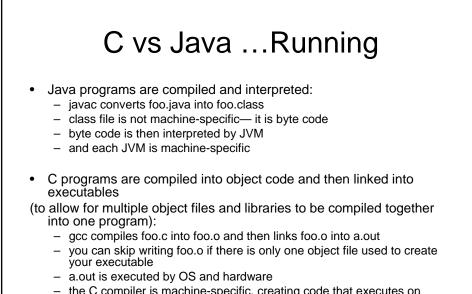
## Background III

- C is early-70s, procedural language
- · C advantages:
  - direct access to OS primitives (system calls)
  - more control over memory
  - fewer library issues- just execute
- C disadvantages:
  - language is portable, but APIs are not
  - no easy graphics interface
  - more control over memory (i.e., memory leaks)
  - pre-processor can lead to obscure errors









the C compiler is machine-specific, creating code that executes on specific OS/hardware

