Operating Systems
EDA092/DIT400

Lab Assignment 1:
Shell Programming

Prajith Ramakrishnan

Based on slides by Bhavishya Goel & Yiannis Nikolakopoulos

Outline

- Logistics
- Shell
- Lab 1: Description & Specifications
- Optional Specifications
- Getting started...
- Parser
- Testing
- Submission

Logistics

- Labs in groups of **2 persons**
  - A groups, B groups – check schedule!
- Lab 1 material accessible **only if** you are a group member!!!
- Finish preparatory assignment in ping pong to access Lab Assignment 1

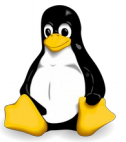
Shell: A Quick Recap

- Definition: a command line interpreter that provides user interface for the operating system
- Basic task:
  - Get input command/s from the user
  - Execute commands and display output
- Note: Shell itself doesn't understand commands (with few exceptions); it only searches for the binary for the given command and executes it with given arguments

Lab 1

- Develop a basic shell program '*lsh*'
- '*lsh*' should be able to replicate the functionality of UNIX shell programs like sh, bash, csh, etc.

Prerequisites for Lab 1

- OS Concepts:
  - Parent and child processes
  - Zombie/Defunct processes
  - Background process
  - UNIX signals and signal-handling
  - System calls like fork, exec, execvp, clone, etc. (for full list, look at lab 1 preparation test questions)
- Basic familiarity with Linux 

Lab 1 Specifications: 1

- Allow users to enter commands to execute programs installed on the system
- *lsh* should be able to execute any binary found in the PATH environment variable
- Example 1: Commands without any argument
  - `'ls'`, `'date'`, `'ps'`, etc.
- Example 2: Commands with arguments
  - `'ls -l'`, `'date -R'`, `'ps aux'`, etc.

Lab 1 Specifications: 2

- Should be able to execute commands in background
- Example:

```
$ sleep 20 &
```

- The '&' sign will spawn the '`sleep`' process in background and the `lsh` will be immediately ready to take next user input

Lab 1 Specifications: 3

- Should support the use of pipes
- Example:

```
$ ls | wc -w
```

  - `ls` outputs the list of all the files and directories in the folder
  - `wc` reads the output from `ls` and counts the number of words in that list
  - `ls` and `wc` communicate using a *pipe*, `ls` writes to the *pipe* and `wc` reads from the *pipe*

Lab 1 Specifications: 4

- Should allow redirection of `stdin` and `stdout` to files
- Example:

```
$ wc -l < /etc/passwd > outfile
```

- The above command creates a new file "outfile" containing the number of accounts on a machine

Lab 1 Specifications: 5 & 6

- `cd` and `exit` are provided as built in functions
- Pressing Ctrl-C should terminate the execution of a program running on your shell, but **not** the execution of the shell itself.
  - Ctrl-C should **not** terminate any background jobs, either.

Optional Specifications

- Add the built in commands `setenv` and `unsetenv` such that one can add and remove environment variables
- Globalising, i.e. one can write

```
$ rm -r *
```

- Job-control
  - Support suspending currently running processes
  - Commands `fg` and `bg` to push processes to background and bring them to foreground respectively

Optional Specifications (contd.)

- Support to write shell scripts

```
$ for i in 1 2 3 4 5
   do
      echo $i
   done
```

  - You are free to create your own syntax

How to get started?

- Finish the preparation test on PingPong
- Download the tarball of source code template from PingPong
  - Basic skeleton
  - Parser to parse command string and prepare command data structure
  - Prints the command entered

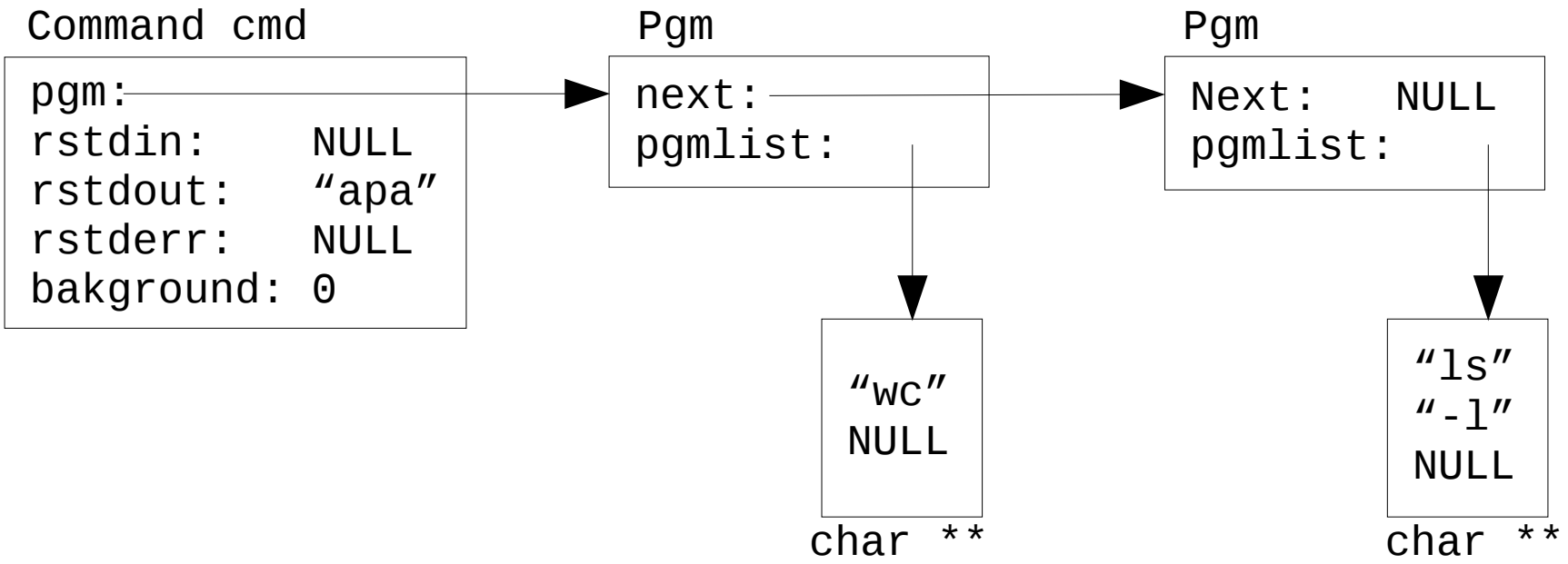
Parser Data Structures

```
typedef struct node {
    Pgm  *pgm;
    char *rstdin;
    char *rstdout;
    char *rstderr;
    int bakground;
} Command;
```

```
typedef struct c {
    char **pgmlist;
    struct c *next;
} Pgm;
```

Parser

- parse("ls -l | wc > apa", &cmd);



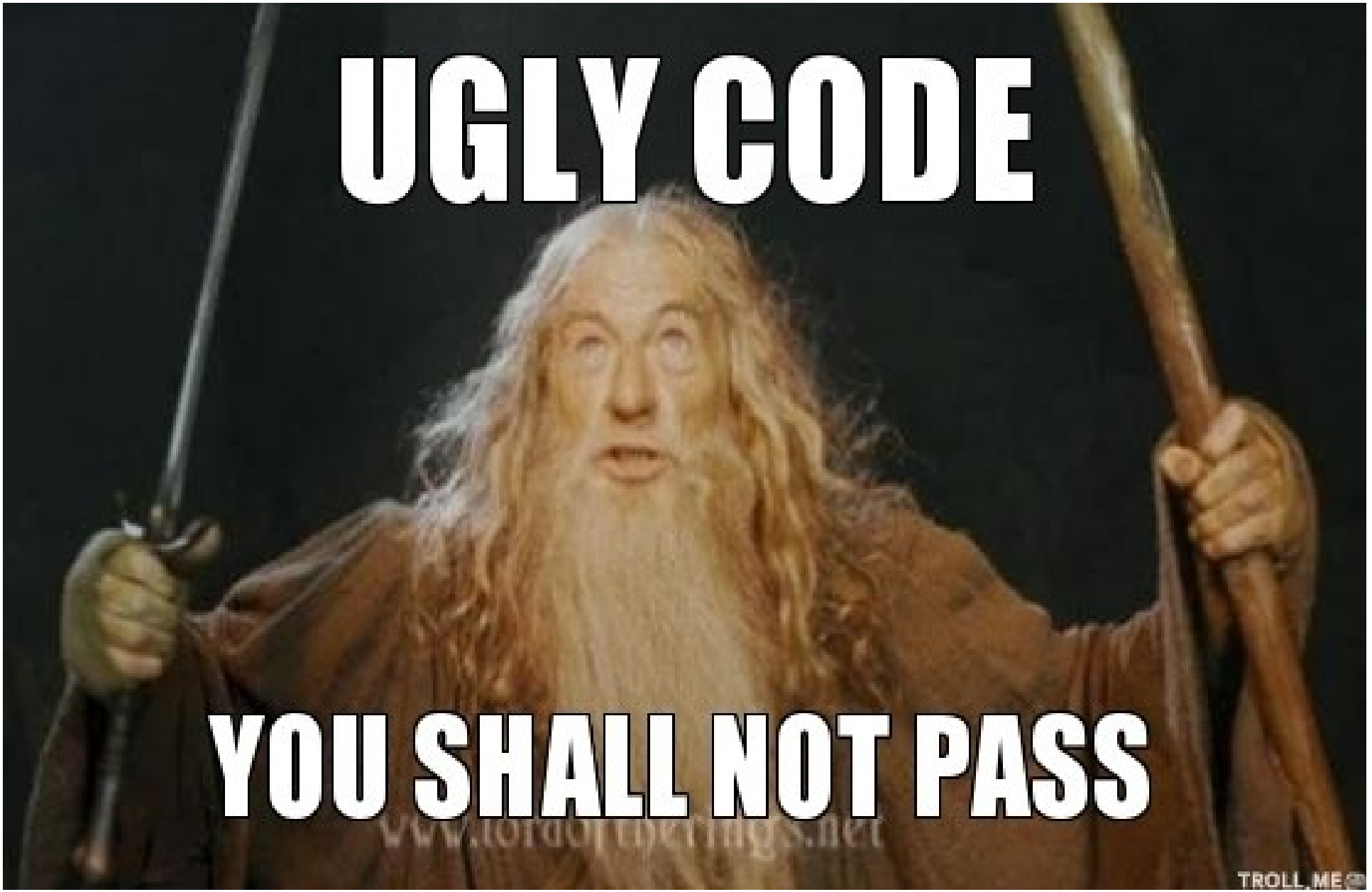
Testing

- Implement and test the specifications one at a time and in the order given
- Use the test commands listed in lab manual
- Be cautious while testing your code remotely as others might also be using the machine at the same time
- Make sure your shell doesn't create zombies at any time

Demo/Submission

- Give the demonstration in the lab
- Submit the (working) code on PingPong

  Deadline: 27th September 2017!
- Code quality:
  - Indentation
  - Comments
  - No debug code (printfs etc)
  - Proper variable names
- Remember...


UGLY CODE
YOU SHALL NOT PASS
TROLL.ME