# Operating Systems
## EDA092/DIT400

## Lab 1 – Discussion

By: Bhavishya Goel (Bhavi)

# Shell: A Quick Recap

- Definition: a command line interpreter that provides user interface for 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

- C programming skills
  - You should know how to handle strings, data structures, pointers, recursive functions, linked lists
- 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 report questionnaire)
- Basic familiarity with simple linux commands.

# 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 argument
  - `'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
```

- The above command outputs the number of files in folder

# Lab 1 Specifications: 4

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

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

- The above command creates a new file "`anatalkonton`" 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. (Preferably **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 as in csh

# Optional Specifications (contd.)

- A real shell language supporting to write shell scripts

```
$ foreach i {1 2 3 4} {
    echo $i
    }
```

# How to get started?

- Complete the preparation report
- Download the tarball of source code template from course webpage
  - Basic skeleton
  - Parser to parse command string and prepare command data structure
  - Prints the command entered

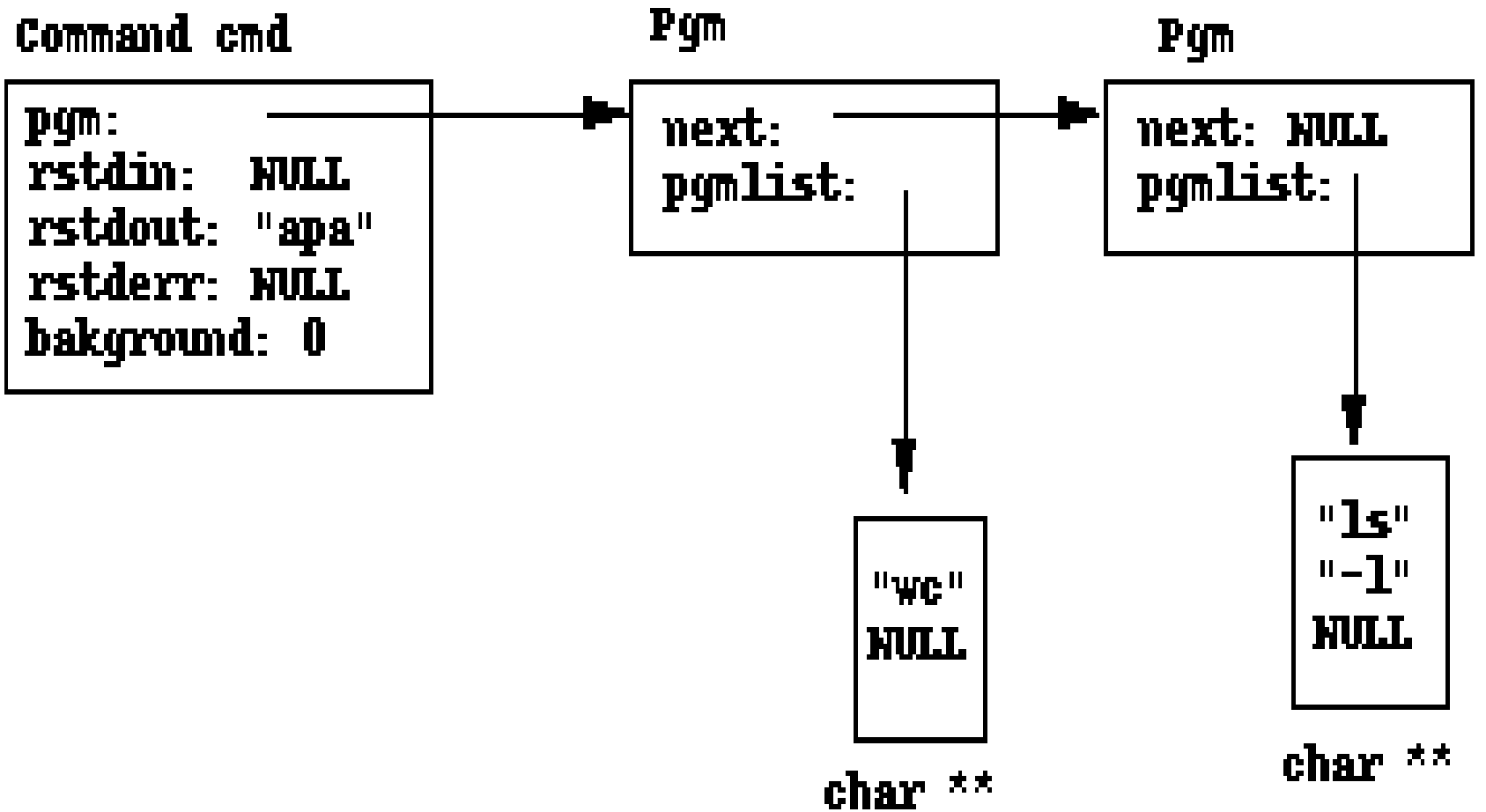
# Parser Data Structures

```
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 );`

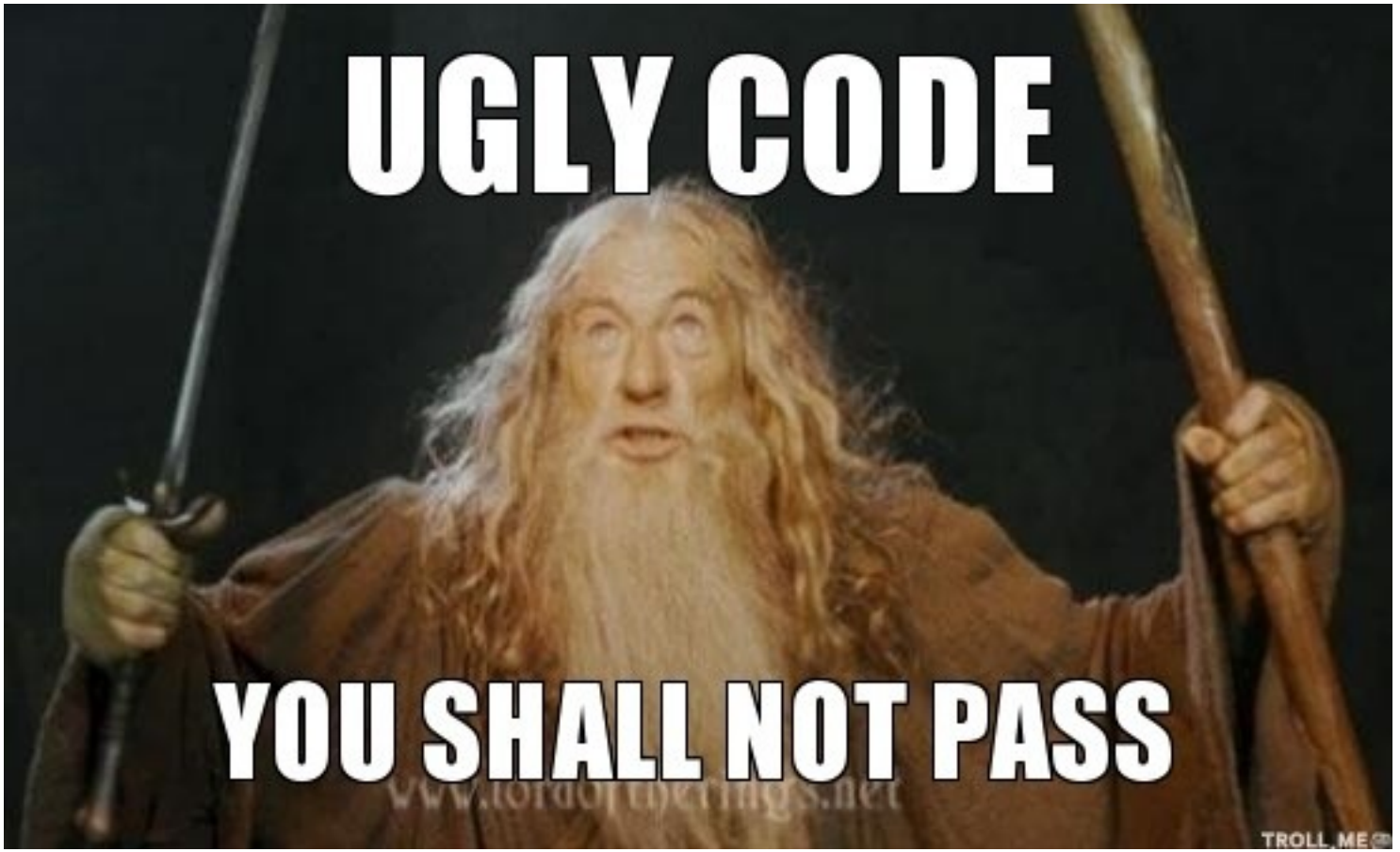


# Testing

- Implement and test the specifications one and a time and in the order given
- Use the test commands listed on the course webpage
- NEVER test your code on a remote server

# Demo/Delivery

- Give the demo in the lab
- Upload the code on Fire
- Code quality:
  - Indentation
  - Comments
  - Clean
  - Proper variable names
- Remember...


UGLY CODE
YOU SHALL NOT PASS
TROLL ME