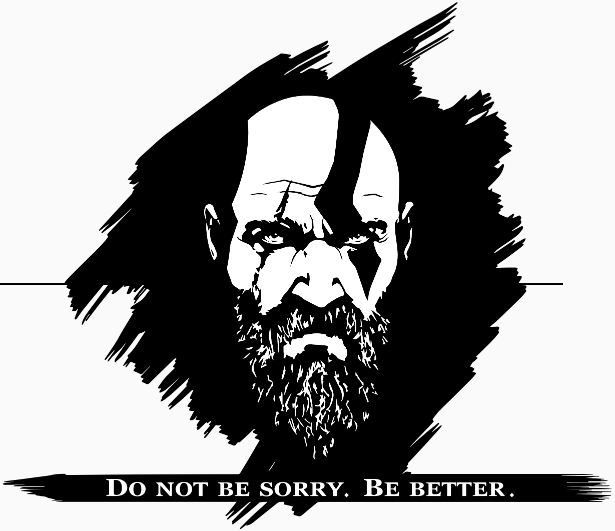
42sh - Presentation

ACU 2019 Team



Copyright

What we have

What we have

```c
void 42sh()
{
    char *input = get_input();
    struct token_list *tokens = lexer(input);
    struct ast_node *ast = parse(tokens);
    execute(ast);
}
```

### The problem

```sh
if ls; then
>
```

### The dirty trick

- Concatenate inputs while the parsing is incomplete
- Pretty slow
- Dirty

# A new lexer

Change architechure

```c
void 42sh()
{
    struct lexer *lexer = new_lexer();
    struct ast_node *ast = parse(lexer);
    execute(ast);
}
```

New lexer

```c
struct lexer
{
    struct token_list *tokens;
};
```

New lexer

```c
enum token_type peek(struct lexer *lexer)
{
    if (lexer->token_list == NULL)
        lexer->token_list = get_tokens(); //readline, then split token
    return lexer->token_list->type;
}

struct token_list *pop(struct lexer *lexer)
{
    if (lexer->token_list == NULL)
        lexer->token_list = get_tokens(); //readline, then split token
    struct token_list *first_token = lexer->token_list;
    lexer->token_list = lexer->token_list->next;
    return first_token;
}
```

Use it

```c
int grammar_rule_if(struct lexer *lexer)
{
    if (peek(lexer) != TOKEN_IF)
        return FALSE;
    free(pop(lexer));
    //parse recursively...
}
```

Why it is better

```
if ls; then
>
```

- After parsing the first line, readline will wait for the new line

Going further

- You may want to look at the stream API
- Implement the same methods may be helpful

### What about context?

```
echo then
```

- Failure

Contextual lexer

Parser call

```
echo then
```

- In this context, `then` is just a `WORD`, not a `THEN_TOKEN`

Grammar

```
simple_command: (prefix)+
    |   (prefix)* (element)+

prefix:     ASSIGMENT_WORD
    |   redirection

element:    WORD
    |   redirection
```

Token type

```
echo then
```

- In this context, `then` is a `WORD`
- If we lex all the line at once, we can’t do the difference

New New Lexer

```c
struct lexer
{
    char *input;  // The whole line
    size_t index; // Start of the next unlexed token
};
```

Use it

```c
struct token *pop(struct lexer *lexer)
{
    if (input == NULL)
        // Readline, set index to 0
    return lex_next_token(lexer); // Lex as usual, but only the first token
}

struct token *pop_command(struct lexer *lexer)
{
    if (input == NULL)
        // Readline, set index to 0
    return lex_next_token_command(lexer); // Lex first token, but don't consider keyword
}
```

In our parser

- In most case, we use `pop()`
- Only when we are sure that there is no keyword (simple_command, ...), we used `pop_command()`

Result

```
echo then
```

- We handle it!

Questions

Questions?