A Teaching Assistant for the C Language

Rui Mendes José João Almeida

Motivation

- Pandemic
- Large classes
- Automatic evaluation
- Need to provide feedback

Problem: If it *works*, it is correct!

A working solution can be

- Poorly written
- Hard to read
- Difficult to maintain
- Poorly documented

How to teach programming?

1. Imperative languages
2. Functional languages
3. Software engineering

Bloom taxonomy

1. Remember
2. Understand
3. Apply
4. Analyse
5. Evaluate
6. Create

A possible solution

- Automatic evaluation of computer programs
- Software metrics
- Documentation coverage

Software metrics

- They help **analyse** and **evaluate** written code
- Complexity
  - Cyclomatic complexity
  - Halstead volume
  - Efective lines of code
- Maintainability
- Can be used to **understand** which are the poorly written functions

Documentation

- Introduces discipline
- Writing documentation forces students to:
  - **Understand** what each function does
  - **Identify** the function of each argument
  - **Understand** what each function returns
  - **Evaluate** their code

Problems with documentation

Students often

- Forget to document arguments
- Confuse arguments with local variables
- Don't understand what is returned
- Forget to update documentation after changing functions
- Existing documentation coverage doesn't check these problems

Query language

```
SHOW <expressions  separated by spaces or semicolons>
[HEADER <fields separated by spaces>]
[COND <Python conditions using  fields>]
[SORT <fields separated by spaces>]
[COLOR <field> : <expression>[; <field> : <expression>]*]
[
GROUP_BY <fields separated by spaces>
[AGGREG <expressions separated by spaces or semicolons>]
]
```

Example

````
# List of problematic functions
These functions are too complex and should be rewritten:

```
SHOW project name cyclomatic_complexity maintainability_index
HEADER project name CC MI
COND CC > 10 or MI < 60
SORT project -MI CC name
COLOR
    CC : scale_lower(0,50)(CC);
    MI : scale_upper(0,100)(MI)
```
````

Example

Problematic functions

Programming 101

Ano letivo 2020/21

folder	name	filename	CC	MI	BP
PL1G02	execute_stack	main.c	22	37.6	2.78
PL1G02	COMMA	main.c	10	56.84	1.37
PL1G02	Load_to_Stack	main.c	12	58.77	1.27
PL1G02	push	main.c	28	61.19	0.83
PL2G01	maior	Code/Módulos/Logica/logica.c	37	52.62	0.86
PL2G01	igual	Code/Módulos/Logica/logica.c	37	52.81	0.86
PL2G01	menor	Code/Módulos/Logica/logica.c	37	52.99	0.87
PL2G01	emaior	Code/Módulos/Logica/logica.c	37	53.05	0.86
PL2G01	emenor	Code/Módulos/Logica/logica.c	37	53.05	0.86
PL2G01	ee	Code/Módulos/Logica/logica.c	37	54.18	1.02
PL2G01	eou	Code/Módulos/Logica/logica.c	37	56.21	0.85
PL2G01	subtrai	Code/Módulos/Math/maths.c	19	63.29	0.77
PL2G01	expoente	Code/Módulos/Math/maths.c	19	63.34	0.82
PL2G01	soma	Code/Módulos/Math/maths.c	19	63.63	0.77
PL2G01	multiplica	Code/Módulos/Math/maths.c	19	64.02	0.76
PL2G01	tokenizador	Code/Parser/tokenizador.c	22	67.53	0.52
PL2G03	parser	parser.c	109	0	6.3
PL2G03	divNumbers	functions.c	17	68.35	0.56
PL2G03	mulNumbers	functions.c	17	68.7	0.56
PL2G03	subNumbers	functions.c	17	68.7	0.56
PL2G03	addNumbers	functions.c	17	69.92	0.57
PL2G05	logicanormal	logica.c	21	69.19	0.55

Figure 1: Listing only problematic functions

Examples using aggregation

```
SHOW name project return loc
GROUP_BY project return
AGGREG len(name) min(loc) mean(loc) max(loc)

SHOW name project return loc
GROUP_BY project
AGGREG mean(loc); (lambda L: len([x for x in L if x < 10]))(loc)
```

Anatomy of a project, part II

- Mooshak for automatic project evaluation
- C teaching assistant for:
  - Flagging function complexity
  - Helping check documentation mistakes
  - Quantitative assessment

Conclusions

- Helps create reports
- Helps students understand:
  - What part of the code should be rewritten
  - What functions are not correctly documented

Availability

https://github.com/rcm/C_teaching_assistant