

# ECOTE - Final Project

Semester: 20L

Author: Sotaro Suzuki 317340

Subject: Simple Macrogenerator – one level of definitions and calls

## I. General Overview and Assumptions

The task is to design a macrogenerator supporting one level of calls and definitions. The macrogenerator also uses named arguments – that means that the arguments must be declared in the macro definition, with a unique name, and then used in the definition body with the correct name. The macrogenerator should support following discriminants: **Discriminant**      **Meaning**

|    |                           |
|----|---------------------------|
| #  | Start of macro definition |
| \$ | Start of macro call       |

With following additional special symbols used in macro definitions and calls:

| Symbol | Meaning                        |
|--------|--------------------------------|
| (      | Start of macro argument list   |
| )      | End of macro argument list     |
| {      | Start of macro definition body |
| }      | End of macro definition body   |
| &      | Macro argument                 |
| ,      | Argument separator             |

It was assumed that, since macro argument names can be longer than one symbol, they are marked by the discriminant ‘&’ at both beginning and end of the name in a macro body.

Therefore, an example macro definition and call would be:

```
#MACRO1(P1, P2)
{hello&P1&macro&P2&world}
$MACRO1(33,44)
```

Which would result in the following text:

```
hello33macro44world
```

Since the macro generator should only support one level of calls and definitions, an assumption was made that using a definition/call inside another definition/call results in an error.

### Whitespace assumptions:

1. Whitespace in macro names is not allowed – therefore whitespace appearing between a ‘#’ and a ‘(’ symbol is treated as an error.
2. Whitespace in parameter names is not allowed – therefore whitespace appearing between a ‘&’ symbol and a ‘,’ or ‘)’ is treated as an error (only in macro definitions)
3. Macro definition argument declarations can be separated by whitespace for visibility, therefore whitespace is allowed between ‘,’ and ‘&’ symbols (but not in macro calls)

4. Macro definition body can be separated from the macro definition name and argument list by whitespace. Therefore, whitespace between ')' and '{' symbols is ignored. For consistency, the whitespace between '}' and the next non-whitespace character is also ignored, as is whitespace between '#' and last preceding non-whitespace character.
5. Whitespace is not ignored in macro call argument lists and macro definition bodies, as they contain free text which may contain whitespace. Therefore, the following are equivalent:

|                    |   |                        |
|--------------------|---|------------------------|
| #M(1, 2){a&1&b&2&} | = | #M(1,2){a&1&b&2&}      |
| #M(1, 2){a&1&b&2&} | = | #M(1, 2)<br>{a&1&b&2&} |

While the following are not:

|                        |   |                           |
|------------------------|---|---------------------------|
| \$M(aa, bb)            | ≠ | \$M(aa,bb)                |
| #M(1, 2)<br>{a&1&b&2&} | ≠ | #M(1, 2)<br>{a &1& b &2&} |

And the following are not allowed (result in errors):

|                      |
|----------------------|
| #MACRO NAME(1, 2)    |
| #M(PARAM 1, PARAM 2) |

## II. Functional Requirements

The general purpose of a macrogenerator is text transformation with the use of macro definitions and calls. The transformation is dynamic, as the macro definitions are provided inside the transformed text, alongside macro calls and free text. This means that a macro library is needed, in which all of the macro definitions encountered in the source text will be stored and then used to resolve macro calls. This data structure is further described in the *Data structures* section. The macrogenerator evaluates each character, changing its behaviour if it encounters a macro definition or call, or another special symbol (depending on context – e.g. argument separator in an argument list). The special symbols of this macrogenerator are:

| Symbol | Meaning                        |
|--------|--------------------------------|
| #      | Macro definition               |
| \$     | Macro call                     |
| (      | Start of macro argument list   |
| )      | End of macro argument list     |
| {      | Start of macro definition body |
| }      | End of macro definition body   |
| &      | Macro argument                 |
| ,      | Argument separator             |
| \      | Escape character               |

The escape character '\' is implemented to allow the use of special characters in free text. Therefore, to output '\' we need '\\' as input. The macrogenerator can also output errors and warnings, if incorrect, unexpected or inadvisable input sequence is encountered. To create useful error/warning outputs, the macrogenerator counts the lines in input text and stores at which line the error or warning occurred,

to then inform the user. The exact types of errors and warnings are described in the *Input/output description* section.

### III. Implementation

#### General Architecture

The program consists of four main modules:

- Main Module – responsible for the Command Line Interface and File I/O
- MacroGenerator Module – responsible for the text transformation
- Error Module – responsible for the list of descriptions of errors and warnings
- Symbol Module – describes the special symbols used by the generator as constants

The class diagram and description is presented below:

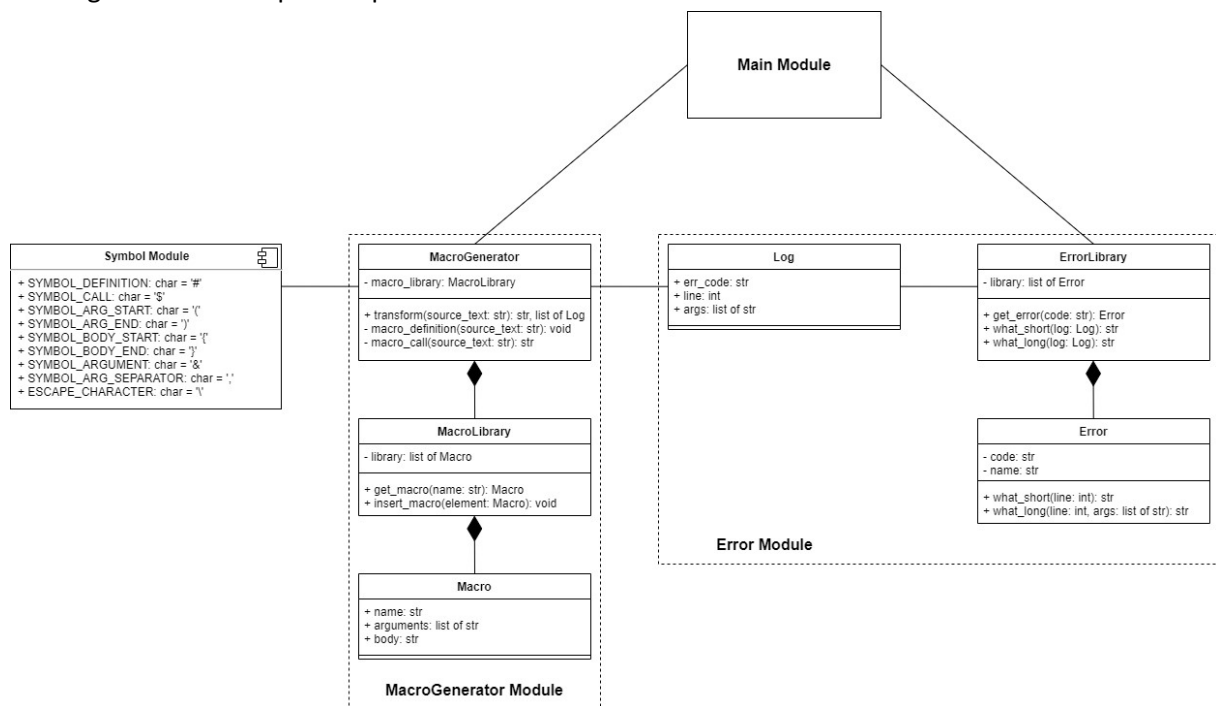


Figure 1: Class Diagram

#### Class MacroGenerator

|  |  |
|--|--|
| <b>macro_library: MacroLibrary</b>                   | MacroLibrary object storing macro definitions  |
| <b>transform(source_text: str): str, list of Log</b> | Main macro generator function. It transforms the source text – when a definition or call is encountered, the appropriate methods are called. It prepares a list of Log objects of all warnings encountered, and in case of an Error throws the Log instead. Then, the output text is returned as string, and a list of warnings. |
| <b>macro_definiton(source_text: str): void</b>       | A method called when a macro definition is encountered. It extracts the name, arguments and body from the source text and inputs, and adds the macro to the library.   |

|  |   |
|--|---|
| <b>macro_call(source_text: str): str</b> | A method called when a macro call is encountered. It extracts the name and arguments, then fetches the macro definition from the library and performs substitution. Returns the substituted text. |
|--|---|

#### Class MacroLibrary

|   |  |
|---|--|
| <b>Library: list of Macro</b>             | A list of Macro objects – macro definitions in the library |
| <b>get_macro(name: str): Macro</b>        | A method returning a Macro object given a macro name.      |
| <b>insert_macro(element: Macro): void</b> | A method inserting a new Macro object into the library.    |

#### Class Macro

|                               |   |
|-------------------------------|---|
| <b>name: str</b>              | The name of the macro                     |
| <b>arguments: list of str</b> | A list of all argument names in the macro |
| <b>body: str</b>              | The macro definition body                 |

#### Class Log

|                          |   |
|--------------------------|---|
| <b>err_code: str</b>     | The error/warning code encountered  |
| <b>line: int</b>         | The line at which the error/warning occurred  |
| <b>args: list of str</b> | List of additional arguments, used to produce the verbose error description, e.g. macro name. |

#### Class ErrorLibrary

|                                    |   |
|------------------------------------|---|
| <b>library: list of Error</b>      | A list of Error objects   |
| <b>get_error(code: str): Error</b> | A method returning an Error object given the error code               |
| <b>what_short(log: Log): str</b>   | A method generating a short error description based on a Log object   |
| <b>what_long(log: Log): str</b>    | A method generating a verbose error description based on a Log object |

#### Class Error

|   |  |
|---|--|
| <b>code: str</b>                                    | The error code                             |
| <b>name: str</b>                                    | The name of the error                      |
| <b>what_short(line: int): str</b>                   | Method returning short error description   |
| <b>what_long(line: int, args: list of str): str</b> | Method returning verbose error description |

## Data Structures

| Structure           | Description  |
|---------------------|--|
| <b>MacroLibrary</b> | List of macro definitions. It consists of Macro objects. As the macro names are unique, it should be a list that allows such behaviour.            |
| <b>ErrorLibrary</b> | List of error and warning definitions. It consists of Error objects. As the error codes are unique, it should be a list that allows such behaviour |

The remaining structures in the program are just simple lists, which can use any implementation.

## Module Descriptions

The two most important parts of the modules are presented below: the execution of the main module, and execution of the MacroGenerator module, which is the module performing the most crucial part of the macro generator operation.

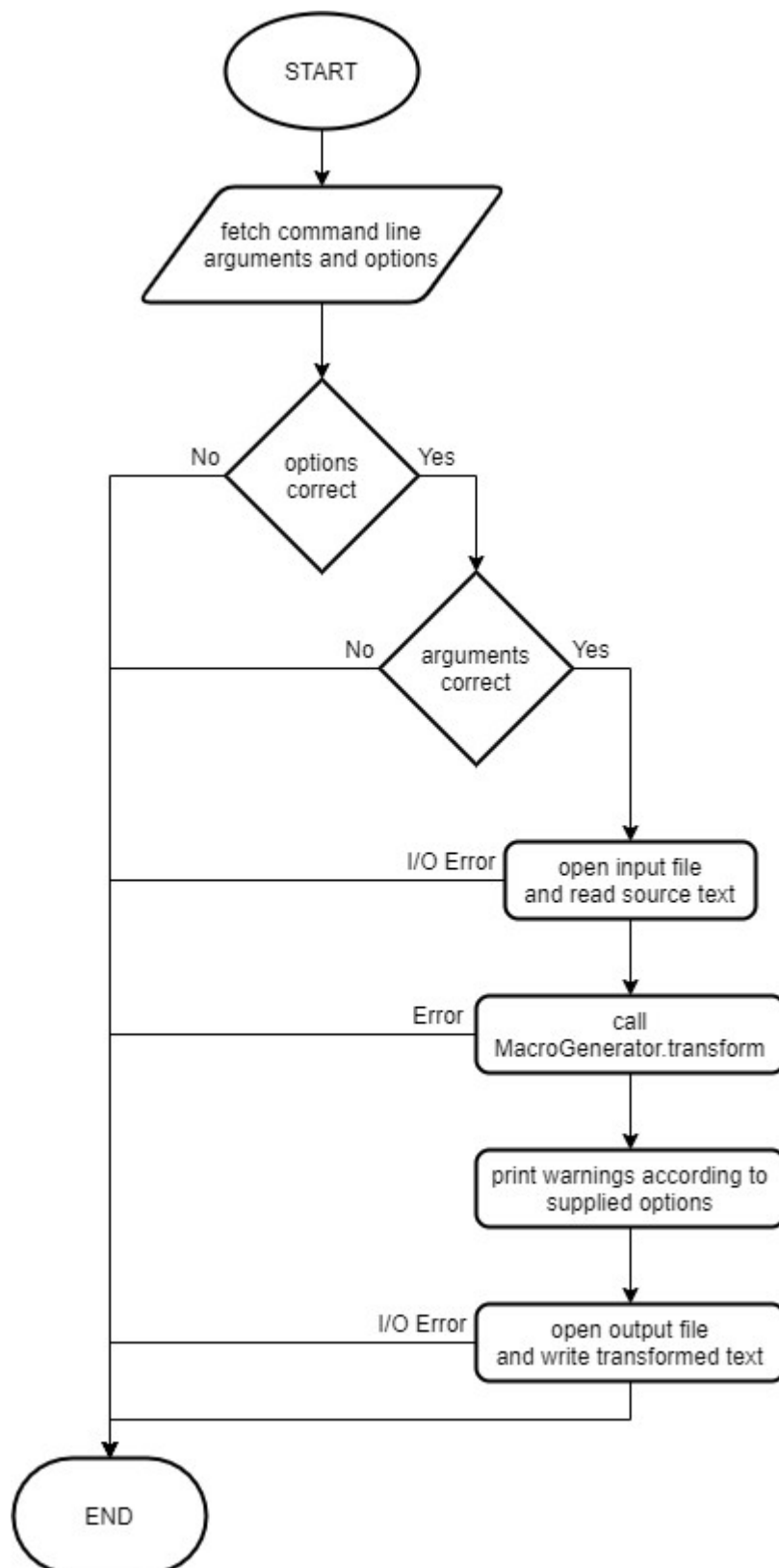
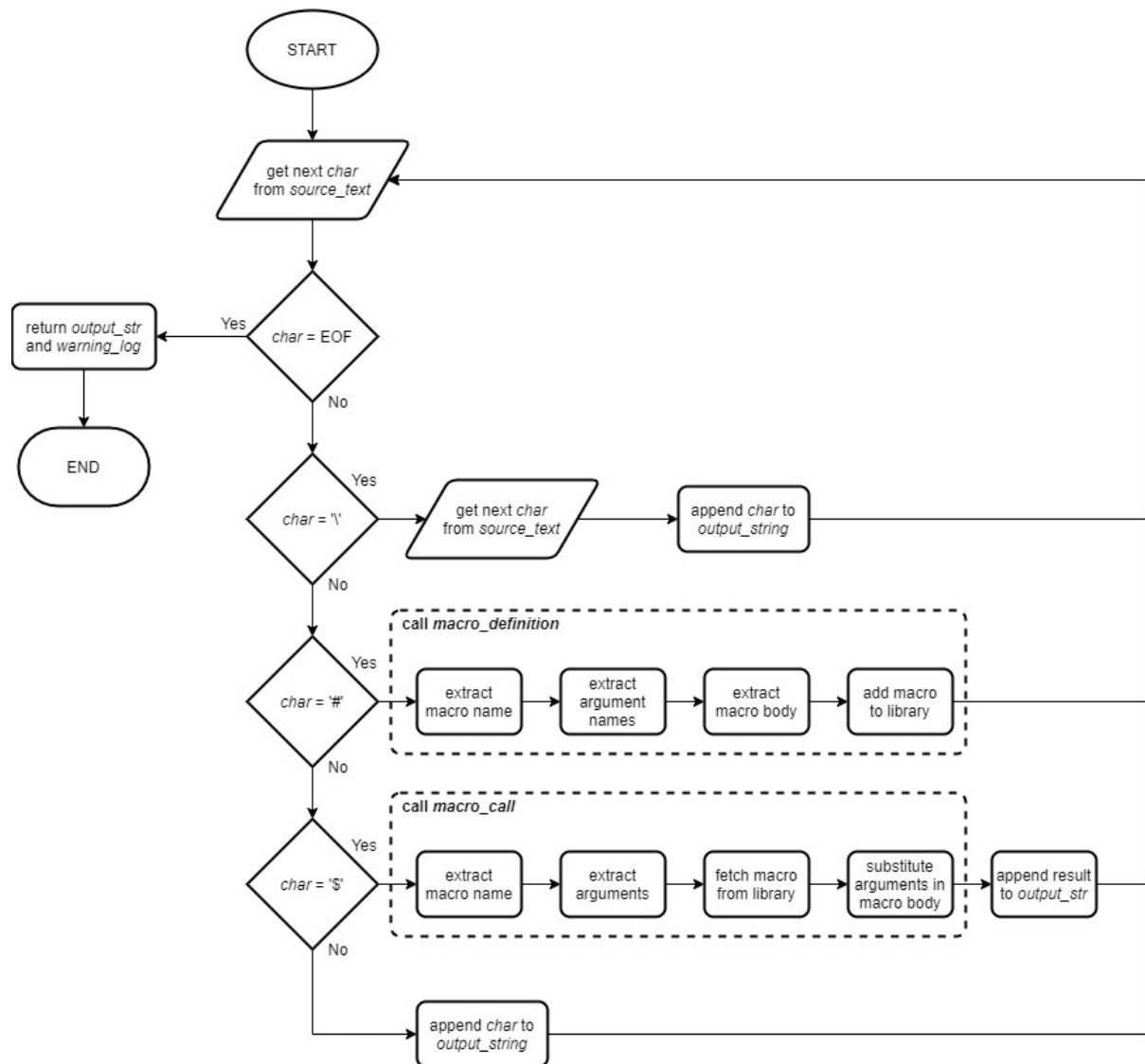


Figure 2: Main Module Flowchart



**Figure 3: MacroGenerator Module Flowchart**

The flowchart for MacroGenerator does not include error handling. At any point in the execution, when an error or a warning occurs, an exception will be thrown, or a warning will be added to the log list.

### Input/Output Description

The program uses a text file as input, and outputs transformed text into another input file. The macro generator transformation itself works as previously described. Below are some basic examples of usage of this macro generator.

#### Input

```
#SIGNATURE(){John Brown}
$SIGNATURE()
```

#### Output

John Brown

#### Example 1: Basic macro with no arguments Input

```
#COPYRIGHT(NAME, YEAR)
{Made by &NAME& in &YEAR&}
$COPYRIGHT(John Brown,2003)
```

#### Output

Made by John Brown in 2003

#### Example 2: Basic macro with two arguments

| Input  | Output                 |
|--|------------------------|
| #SIGNATURE(){John Brown}<br>#COPYRIGHT(NAME, YEAR)<br>{Made by &NAME& in &YEAR&}<br>\$COPYRIGHT(\$SIGNATURE, 2003) | Error e23: Nested Call |

**Example 3: Incorrect usage - nested call** (more on errors in the Errors and Warnings section)

| Input  | Output                   |
|--|--------------------------|
| #TOTAL(PRICE)<br>{The total price is \&PRICE&}<br>\$TOTAL(749) | The total price is \$750 |

**Example 4: Usage of the escape character '\'**

### Command Line Interface

The program is operated with a command line interface (CLI). It supports a few options regarding the error/warning output. The basic use case for this CLI is as follows:

```
>mg input_file output_file
```

The program takes 'input\_file' as the source file with text to be transformed and outputs the transformed text into 'output\_file'. The errors and warnings are outputted to stdout. If the program is run with only one argument, it uses it as the input, and outputs to a text file called 'mg\_out' instead. Running the program with no arguments or too many arguments results in an error, and it is not executed. The program also supports options to control the error and warning output, that can be inserted before the I/O files. The option summary can be seen in the table below.

| Option    | Short version | Arguments | Description   |
|-----------|---------------|-----------|---|
| --silent  | -s            | None      | Mutes the error and warning output  |
| --nowarn  | -n            | None      | Mutes the warning output  |
| --output  | -o            | Filename  | Redirects the error and warning output to a file.                                   |
| --verbose | -v            | None      | Makes the error and warning output more verbose to give a clearer error information |

Choosing a non-existent option or incompatible options (like -s + -v) will result in an error and the program will abort execution.

### Errors and Warnings

The macrogenerator program can encounter some errors and warnings when transforming text, as a result of incorrect or unexpected input. In case of an error, the program informs the user of said error and stops execution. In case of a warning the program informs the user and continues text transformation. The way of informing the user can be controlled via the CLI, described further in the *Command Line Interface* section. Errors and warnings are divided into categories, and each of them is given a distinct code. The errors and warnings are presented in the tables below.



| <b>Category</b>               | <b>Code</b> | <b>Error Name</b>        | <b>Description</b>  |
|-------------------------------|-------------|--------------------------|---|
| <i>Macro Definition Error</i> | e10         | Incorrect Macro Name     | The macro name in a macro definition contains an incorrect character – whitespace or one of the special characters (#,\$,(,),{,&,\)     |
|                               | e11         | Macro Already Defined    | The macro definition defines a macro with a name that already exists in the name library  |
|                               | e12         | Incorrect Parameter Name | The parameter name in a macro definition contains an incorrect character – whitespace or one of the special characters (#,\$,(,),{,&,\) |
|                               | e13         | Missing Macro Body       | A non-whitespace character was encountered between ‘)’ and ‘{’ in a macro definition  |
|                               | e14         | Parameter Undefined      | A parameter name used in the macro body is not present in the argument list   |
|                               | e15         | Nested Call              | A macro call was encountered inside a macro body  |
|                               | e16         | Nested Definition        | A macro definition was encountered inside a macro definition  |
|                               | e17         | Parameter Repeated       | The same parameter was used twice or more in a macro definition   |
|                               | e18         | Unfinished Definition    | The file stream ended inside a macro definition   |
| <i>Macro Call Error</i>       | e20         | Undefined Macro          | A macro name called was not found in the macro library  |
|                               | e21         | Too Few Arguments        | Macro was called with less arguments than in the definition   |
|                               | e22         | Incorrect Macro Call     | Macro name that was called contains an incorrect character – whitespace or one of the special characters (#,\$,(,),{,&,\)               |
|                               | e23         | Nested Call              | A macro call was encountered inside a macro call  |
|                               | e24         | Nested Definition        | A macro definition was encountered inside a macro call  |
|                               | e25         | Unfinished Call          | The file stream ended while inside a macro call   |
| <i>Other</i>                  | e98         | I/O Error                | There was an error with file I/O  |

| Category                 | Code | Warning Name           | Description   |
|--------------------------|------|------------------------|---|
| Macro Definition Warning | w10  | Unused Parameter       | Parameter from the definition is not used inside the macro body             |
|                          | w11  | Empty Macro Body       | The macro body was left empty   |
|                          | w12  | Unused Macro           | Macro definition was never called   |
| Macro Call Warning       | w20  | Too Many Arguments     | The macro was called with more arguments than the definition                |
|                          | w21  | Empty Argument         | The argument supplied to the call was empty – nothing between two ‘         |
|                          | w22  | Whitespace Argument    | The argument supplied starts with whitespace – possible error from the user |
| CLI Warning              | w80  | Overwrite Warning      | The input file is the same as the output file                               |
| Other                    | w90  | Escape Character Error | The escape character ‘\’ was used on a non-special character                |

#### IV. Functional Test Cases

The following test cases check all the warnings and errors available in the MacroGenerator and a few correct usage cases. The tests don’t include the CLI tests.

##### Correct Usage

| Test Case                            | Input  | Expected Output                            |
|--------------------------------------|--|--|
| Basic single macro with no arguments | #MACRO(){test macro}<br>\$MACRO()  | test macro                                 |
| Basic single macro with arguments    | #MACRO(P1, P2)<br>{&P1&+&P2*&P1&}<br>\$MACRO(34,20)  | 34+20*34                                   |
| Multiple basic macros                | #MACRO1(){test macro}<br>#MACRO2(P1, P2)<br>{&P1&+&P2*&P1&}<br>\$MACRO2(34,20)<br>\$MACRO1()                           | 34+20*34 test macro                        |
| Multiple macros with free text       | free text<br>#MACRO1(){test macro}<br>#MACRO2(P1, P2)<br>{&P1&+&P2*&P1&}<br>\$MACRO2(34,20)<br>\$MACRO1()<br>MORE TEXT | free text<br>34+20*34 test macro MORE TEXT |
| Escape symbol in free text           | Price: /\$20   | Price: \$20                                |
| Escape symbol in macro body          | #MACRO(A){B\&&A&}<br>\$MACRO(C)  | B&C  |
| Escape symbol in argument            | #MACRO(A){hello &A&}<br>\$MACRO(big \\$)   | hello big \$                               |

##### Macro Definition Errors

| Test Case | Input | Expected Output |
|-----------|-------|-----------------|
|-----------|-------|-----------------|

|                          |  |           |
|--------------------------|--|-----------|
| Incorrect Macro Name     | #MY MACRO(){test macro}                      | Error e10 |
| Macro Already Defined    | #MACRO(P1, P2){&P1&+&P2&}<br>#MACRO(){hello} | Error e11 |
| Incorrect Parameter Name | #MACRO(MY PARAM)<br>{&MY PARAM& is wrong}    | Error e12 |
| Missing Macro Body       | #MACRO()eee{test macro}                      | Error e13 |
| Parameter Undefined      | #MACRO(P1, P2)<br>{&P1& is less than &P3&}   | Error e14 |
| Nested Call              | #HI(){hello}<br>#MACRO(ARG){\$HI &ARG&}      | Error e15 |
| Nested Definition        | #MACRO(){#NEST(){no}\$NEST}                  | Error e16 |
| Parameter Repeated       | #MACRO(A,A){}                                | Error e17 |
| Unfinished Definition    | #MACRO(                                      | Error e18 |

### Macro Call Errors

| Test Case            | Input  | Expected Output |
|----------------------|--|-----------------|
| Undefined Macro      | \$MACRO(hello)                               | Error e20       |
| Too Few Arguments    | #MACRO(P1, P2){&P1&+&P2&}<br>\$MACRO(23)     | Error e21       |
| Incorrect Macro Call | #MYMACRO(P){&P& is good}<br>\$MY MACRO(John) | Error e22       |
| Nested Call          | #A(P){hello &P&}<br>#B(){John}<br>\$A(\$B)   | Error e23       |
| Nested Definition    | #A(P){hello &P&}<br>\$A(#B(){John})          | Error e24       |
| Unfinished Call      | #A(P){hello &P&}<br>\$A(                     | Error e25       |

### Macro Definition Warnings

| Test Case        | Input                                | Expected Output           |
|------------------|--------------------------------------|---------------------------|
| Unused Parameter | #MACRO(P){test macro}<br>\$MACRO(11) | test macro<br>Warning w10 |
| Empty Macro Body | #MACRO(){}<br>\$MACRO()              | Warning w11               |
| Unused Macro     | #MACRO(P){test macro &P&}            | Warning w12               |

### Macro Call Warnings

| Test Case          | Input                                      | Expected Output           |
|--------------------|--|---------------------------|
| Too Many Arguments | #MACRO(){test macro}<br>\$MACRO(John)      | test macro<br>Warning w20 |
| Empty Argument     | #MACRO(P1, P2){&P1&+&P2&}<br>\$MACRO(, 34) | +34<br>Warning w21        |

|                     |   |                             |
|---------------------|---|-----------------------------|
| Whitespace Argument | #MACRO(P1, P2){&P1&+&P2&}<br>\$MACRO(34, 2) | 32+ 2<br><i>Warning w22</i> |
|---------------------|---|-----------------------------|

## V. Python Implementation

The program was implemented using **Python**, version **3.6.9**, and ran and tested under Linux. The macrogenerator part of the program uses the standard Python data structure, *list*. The main module, responsible for the CLI and File I/O uses the Python libraries *sys* and *optparse*. For testing the library *unittest* was used. The macrogenerator and error modules can be imported and used outside from the supplied CLI application. The source code is available in the *src* directory, inside which are *main.py*, the main CLI application, *test.py*, which runs the tests, and separate folders with the *Error*, *Macrogenerator* and *Symbol* modules. The implementation is available in a repository on github:

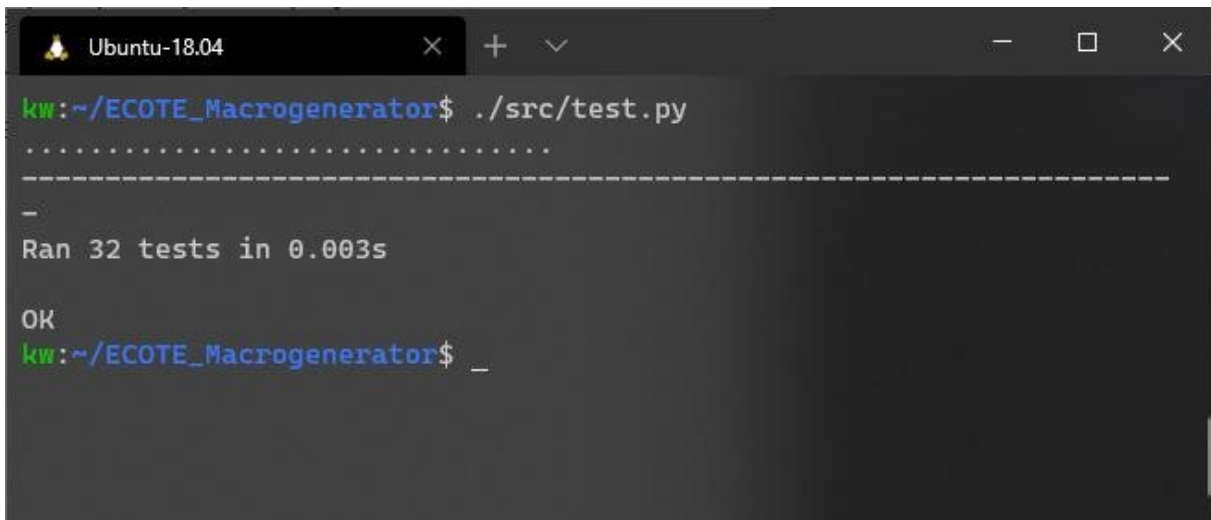
[https://github.com/kacpwoja/ECOTE\\_Macrogenerator](https://github.com/kacpwoja/ECOTE_Macrogenerator)

### File Structure

- *example\_input* file with example input for the program
- *src* directory with source code
  - *main.py* main CLI program
  - *test.py* script running tests
  - *symbol* directory with the symbol module
    - *\_\_init\_\_.py*
    - *symbol.py* definitions of symbols
  - *error* directory with the error module
    - *\_\_init\_\_.py*
    - *error.py* the Error class
    - *log.py* the Log class
    - *errorlibrary.py* the ErrorLibrary class
  - *macrogenerator* directory with the macrogenerator module
    - *\_\_init\_\_.py*
    - *macro.py* the Macro class
    - *macrolibrary.py* the MacroLibrary class
    - *macrogenerator.py* the MacroGenerator class
    - *test\_macrolibrary.py*
    - *test\_macrogenerator.py*

### Running the program

The tests can be run by running the *src/test.py* file.

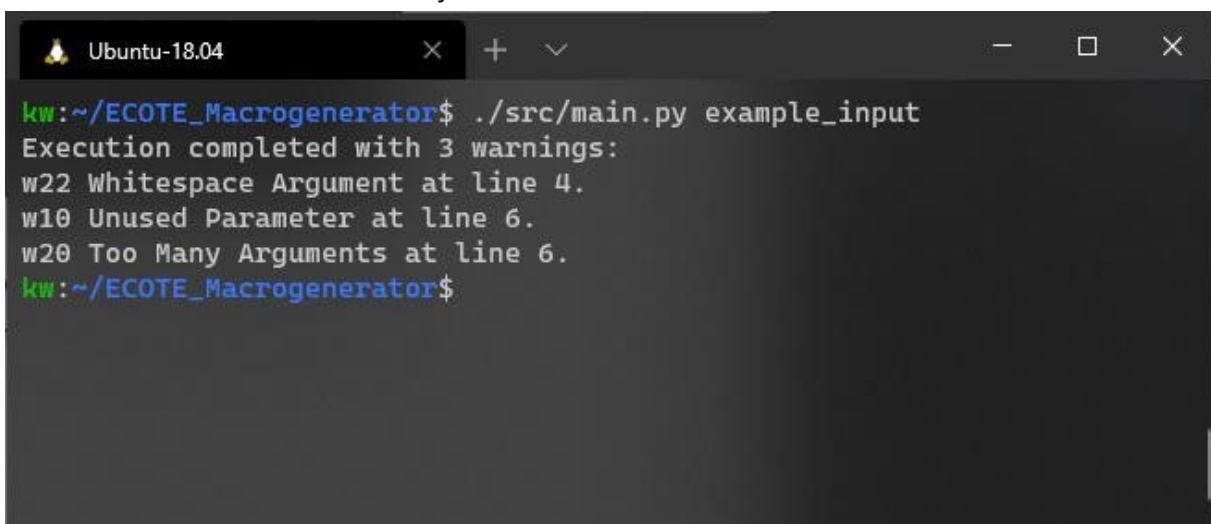
A terminal window titled 'Ubuntu-18.04' with standard window controls. The prompt is 'kw:~/ECOTE\_Macrogenerator\$'. The command './src/test.py' has been executed. The output shows a series of dots, a dashed line, a hyphen, and then 'Ran 32 tests in 0.003s' followed by 'OK'. The prompt is now 'kw:~/ECOTE\_Macrogenerator\$ \_'.

```
kw:~/ECOTE_Macrogenerator$ ./src/test.py
.....
-----
-
Ran 32 tests in 0.003s

OK
kw:~/ECOTE_Macrogenerator$ _
```

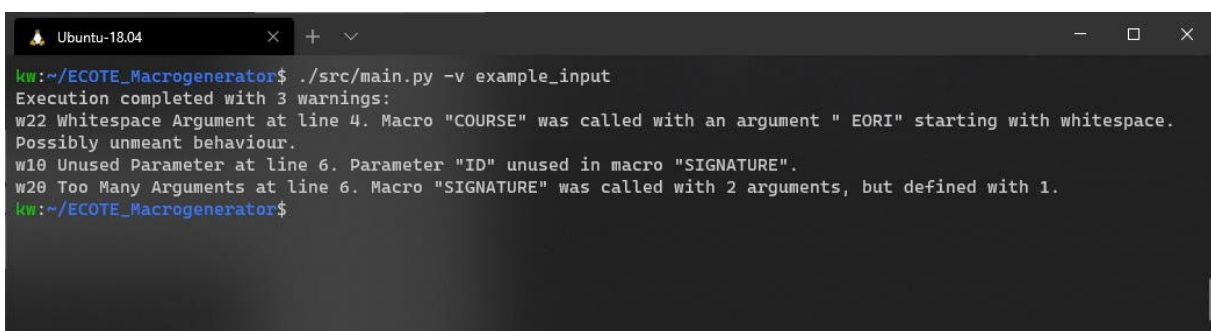
Figure 4: Running the unit tests

The main program is run by running the `src/main.py` file, providing an input file as an argument. A text file `example_input` is provided, which can be used to run the program. Detailed description of the CLI is available in the *Command Line Interface* section.

A terminal window titled 'Ubuntu-18.04' with standard window controls. The prompt is 'kw:~/ECOTE\_Macrogenerator\$'. The command './src/main.py example\_input' has been executed. The output shows 'Execution completed with 3 warnings:' followed by three lines of warnings: 'w22 Whitespace Argument at line 4.', 'w10 Unused Parameter at line 6.', and 'w20 Too Many Arguments at line 6.'. The prompt is now 'kw:~/ECOTE\_Macrogenerator\$'.

```
kw:~/ECOTE_Macrogenerator$ ./src/main.py example_input
Execution completed with 3 warnings:
w22 Whitespace Argument at line 4.
w10 Unused Parameter at line 6.
w20 Too Many Arguments at line 6.
kw:~/ECOTE_Macrogenerator$
```

Figure 5: Example of running the program

A terminal window titled 'Ubuntu-18.04' with standard window controls. The prompt is 'kw:~/ECOTE\_Macrogenerator\$'. The command './src/main.py -v example\_input' has been executed. The output shows 'Execution completed with 3 warnings:' followed by three lines of verbose warnings: 'w22 Whitespace Argument at line 4. Macro "COURSE" was called with an argument " EORI" starting with whitespace. Possibly unmeant behaviour.', 'w10 Unused Parameter at line 6. Parameter "ID" unused in macro "SIGNATURE".', and 'w20 Too Many Arguments at line 6. Macro "SIGNATURE" was called with 2 arguments, but defined with 1.'. The prompt is now 'kw:~/ECOTE\_Macrogenerator\$'.

```
kw:~/ECOTE_Macrogenerator$ ./src/main.py -v example_input
Execution completed with 3 warnings:
w22 Whitespace Argument at line 4. Macro "COURSE" was called with an argument " EORI" starting with whitespace.
Possibly unmeant behaviour.
w10 Unused Parameter at line 6. Parameter "ID" unused in macro "SIGNATURE".
w20 Too Many Arguments at line 6. Macro "SIGNATURE" was called with 2 arguments, but defined with 1.
kw:~/ECOTE_Macrogenerator$
```

Figure 6: Example of verbose warning outputs