ECOTE – Final Project

Date: 04.05.2021

Semester: 21L

Author and Group: Jakub Wojcieszuk Group: 101

Subject: MacroG4

I. General overview and assumptions

The task of the program is to design a macrogenerator with positional parameters in which macro calls and macro definitions can be nested. Positional parameter denominator is $ E.g. $1 – for the substitution of the first parameter. Program handles up to 10 parameters, from $0 up to $9. In order to call a macro which has positional parameters defined, in macro call there should be provided at least same amount of actual parameters. Redundant parameters are ignored.

In the program following strings are defined as special:

String	Meaning
#MDEF	Start of macro definition
#MEND	End of macro definition
#MCALL	Start of macro call

Therefore, an example of macro definition with one positional parameter and macro call would be:

Input	**Output**
#MDEF A **free $0 text A** **#MEND** **#MCALL A argA**	free argA text A

Syntax of the input file:

Macro call syntax

- ‘#MCALL’ should be provided at the beginning of the line
- Macro call syntax: ‘#MCALL name list of current parameters;’
- Macro called should be available in the macro library.
- Each current parameter is separated by a ‘,’ character
- In case if there are positional parameters in macro definition, macro call should include at least the same number of current parameters. The list of current

parameters may be longer than number of positional parameters defined, but redundant parameters will be ignored.
- It is possible that macro call has nested macro call passed as actual parameter. E.g. #MCALL A x,y,#MCALL B; is possible.

Macro definition syntax
- '#MDEF' string should be provided at the beginning of the line.
- Once '#MDEF' was provided correctly, next string separated by blank space is read as a macro name.
- Macro name consists only of letters 'Aa-Zz', without any special characters. Macro name can also be a result of a macro call, so for example '#MDEF #MCALL abc A,B' is also possible. In case of any other strings than #MCALL or 'Aa-Zz' found after #MDEF will result in warning printed to console.
- Macro name is read until new line character is found, or a blank space is found. In case if there are any strings in the same line they will be ignored. E.g '#MDEF name somestring'.
- Macro definition is read from next line after '#MDEF name' is provided and until '#MEND' is found. In case if 'EOF' is reached, and '#MEND' wasn't found, program returns a warning.
- Inside macro definition, there can be nested macro definitions and macro calls.

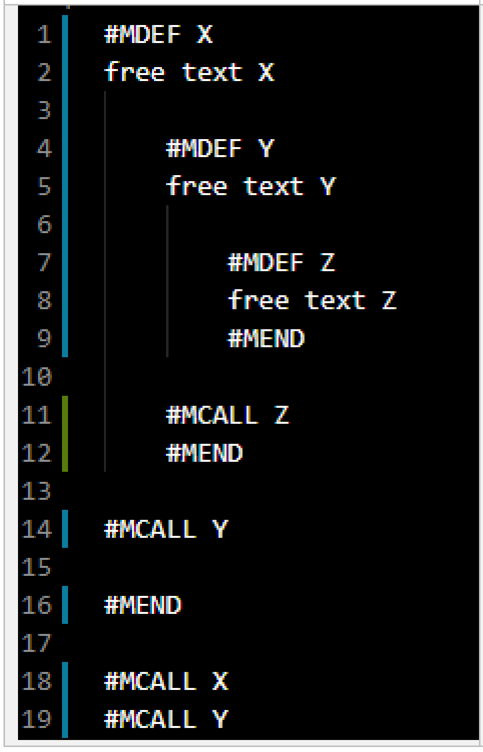
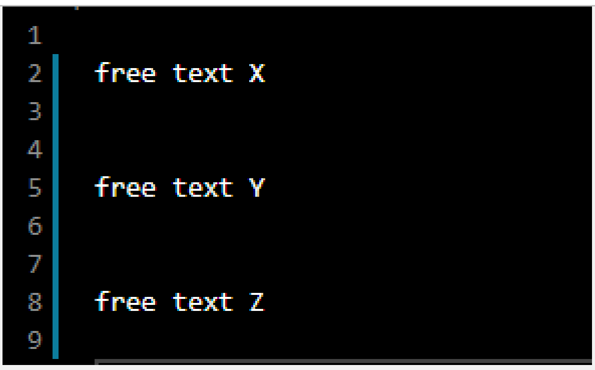
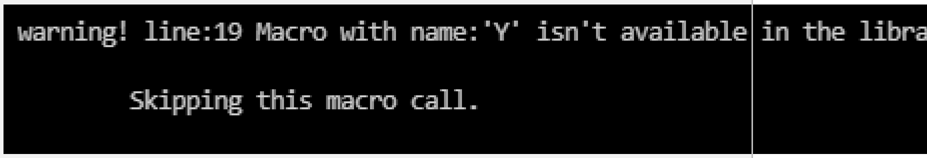
Positional parameters syntax
- In case if '$' is found inside macrodefinition, next character should be a digit. E.g. '$0' – for the substitution of the first parameter. In case if following character is not a digit '0-9' program will print an error to output file. After digit, any following character will be treated as a free text.
- Maximum 9 parameters are allowed, from '$0' up to '$9'
- Positional parameter can be either at the beginning of the line or somewhere inside.

#MEND syntax
- '#MEND' should also be provided at the beginning of the line.

II. Functional requirements
The general purpose of a macrogenerator is text transformation with the use of macro definitions and macro calls. The transformation is dynamic, which means that macro definitions are provided inside transformed text, together with macro calls and free text. In this program it is handled by the use of macro library. Each time macro definition with correct syntax is given, program adds this macro to library and it can be called further in the file. Macrogenerator processes each character and change its behavior once correct special string or symbol is encountered. Furthermore, macrogenerator handles errors and warnings. Each time incorrect syntax is given in the input file, warning or error is printed to the console

together with line number at which it was encountered and a message describing the type of error.

Macro scope handling:
Because of the fact that each macro defintion can be defined in another macro, it is only possible to call a macro visible in current scope.
So for example such input will result in a warning:

Input	**Output**
	 Console: 

```
 1  #MDEF X
 2  free text X
 3
 4      #MDEF Y
 5      free text Y
 6
 7          #MDEF Z
 8          free text Z
 9          #MEND
10
11      #MCALL Z
12      #MEND
13
14  #MCALL Y
15
16  #MEND
17
18  #MCALL X
19  #MCALL Y
```

```
1
2   free text X
3
4
5   free text Y
6
7
8   free text Z
9
```

Console:

```
warning! line:19 Macro with name:'Y' isn't available in the libra

        Skipping this macro call.
```

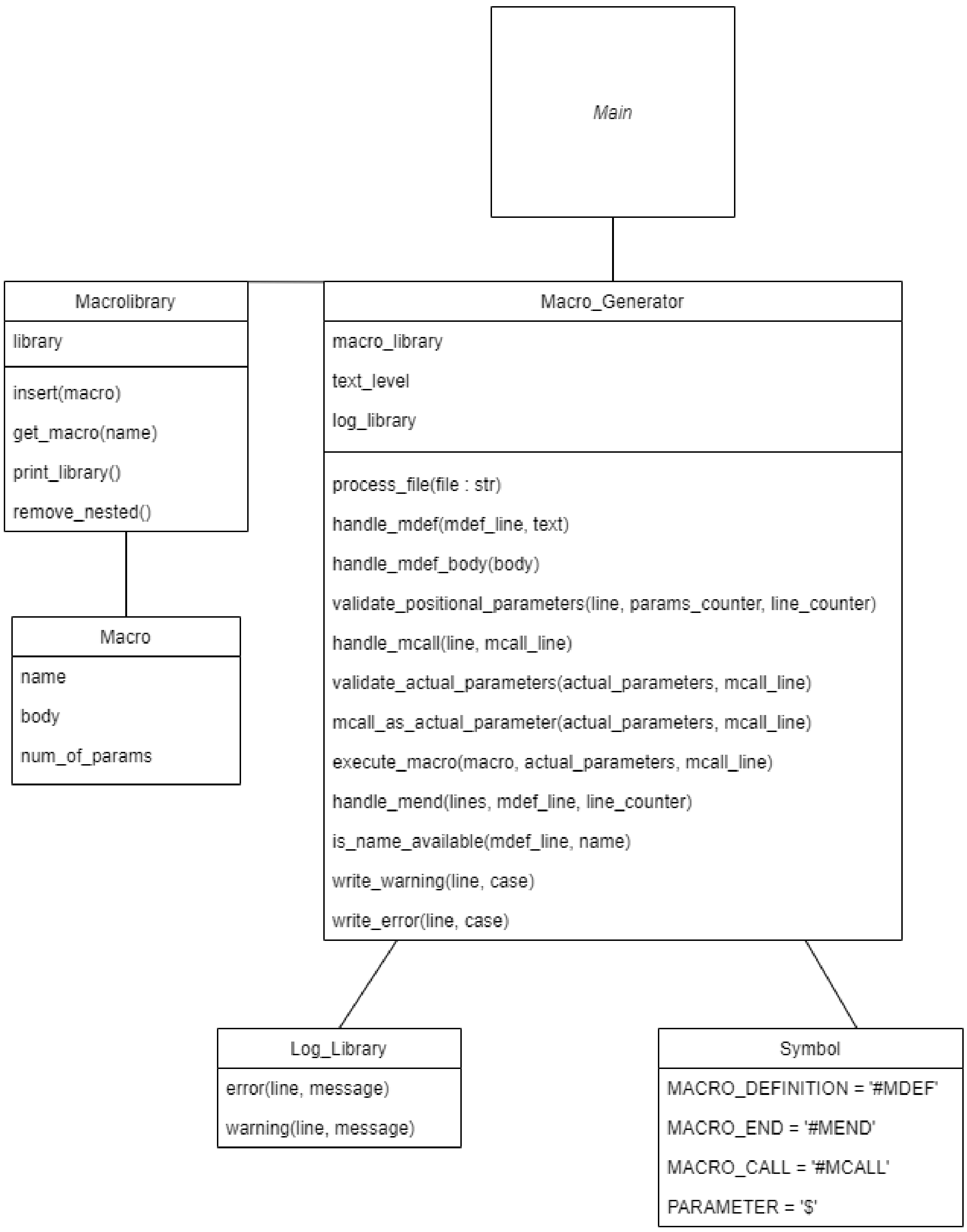
III. Implementation

General architecture:

Program consists of 4 modules.

- Main module, which is responsible for Command Line Interface
- Macro_Generator module, which is responsible for the text transformation
- Symbol module, which defines the symbol used by macrogenerator
- Log module, which defines the list of errors and warnings

Class diagram:


Main
Macrolibrary
library
insert(macro)
get_macro(name)
print_library()
remove_nested()
Macro
name
body
num_of_params
Macro_Generator
macro_library
text_level
log_library
process_file(file : str)
handle_mdef(mdef_line, text)
handle_mdef_body(body)
validate_positional_parameters(line, params_counter, line_counter)
handle_mcall(line, mcall_line)
validate_actual_parameters(actual_parameters, mcall_line)
mcall_as_actual_parameter(actual_parameters, mcall_line)
execute_macro(macro, actual_parameters, mcall_line)
handle_mend(lines, mdef_line, line_counter)
is_name_available(mdef_line, name)
write_warning(line, case)
write_error(line, case)
Log_Library
error(line, message)
warning(line, message)
Symbol
MACRO_DEFINITION = '#MDEF'
MACRO_END = '#MEND'
MACRO_CALL = '#MCALL'
PARAMETER = '$'

Macro_generator class functions:

Function name	Description
process_file(file : str)	Function for processing every single line of input file. Once any special symbol is encountered, proper function is called. Reads from input file and writes to output file, and errors and warnings to console.
handle_mdef(mdef_line : int, macro_text : str)	Function used to handle input text, when #MDEF is encountered. It validates syntax of macro name and as a result inserts macro to library.
handle_positional_parameters(macro_body : str)	Function used to handle positional parameters. It validates if syntax of positional parameter is correct and counts number of positional parameters in a macro.
validate_positional_parameters(line : str, parameters_counter : int, line_counter : int)	Function used inside 'handle_positional_parameters' function to validate positional parameters.
handle_mcall(line : str, mcall_line : int)	Function used to handle input text, when #MCALL is encountered. As input takes line with #MCALL in the beginning. If macro called was found in library, executes it, if not prints an warning on this line to the console.
validate_actual_parameters(actual_parameters : str, mcall_line : int)	Function used in 'handle_mcall' function. It is used to validate syntax of actual parameters.
mcall_as_actual_parameter(actual_parameters : str, mcall_line : int)	Function used when #MCALL is used as a actual parameter. If syntax is correct, executes macro called and replaces its output in place of this call.
execute_macro(macro : macro, actual_parameters : str, mcall_line : int)	Function called when #MCALL is used and macro is found in library. Returns output of macro call.
handle_mend(lines : str, mdef_line : int, line_counter : int)	Function used to handle input text, when #MEND is encountered. If text_level is smaller than 1, prints warning to output. Otherwise calls handle_mdef function.
is_name_available(mdef_line : int, name : str)	Function used to prevent macro definition overwriting. Returns False if

	macro is already defined.

Macro_library class functions:

append(macro)	**Function used to append macro to library.**
get_macro(name : str)	Function returns macro if found in library.
print_library()	Function prints all macros stored in library.
remove_nested()	Function used to remove nested macro definitions, so that the scope processed is handled correctly.

Macro class:

name : str	**The name of the macro**
body : str	The macro definition body
num_of_params : int	Number of parameters in macro definition body

Main assumptions of program algorithm:

1. Check if file given exists and is not empty
2. Read each line one by one:

   Case (**#MDEF**) was given correctly:
   In such case, mdef_line is set as current line and text_level is increased by 1, then loop continues with processing next line.

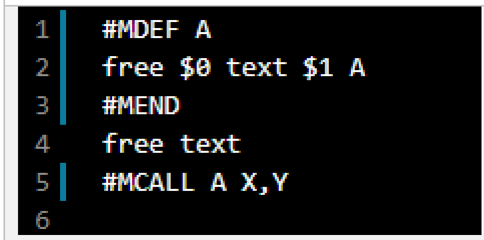
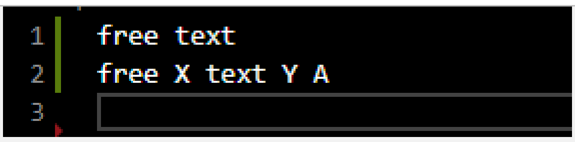
   Case (**#MEND)** was given correctly:
   Firstly, if text_level is > 1, it is decreased by 1 and loop continues with processing next line. If it is < 1, I check whether mdef_line was set previously, if so, program goes to function which is responsible for handling macro definition.

   Case (**#MCALL)** was given correctly:
   Firstly, text_level is checked whether its equal 0, if so, program goes to function which handles mcall and executes it if syntax is correct.

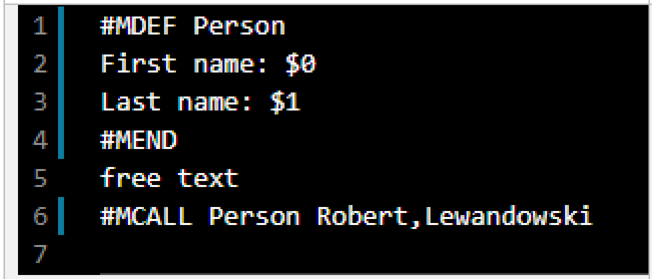
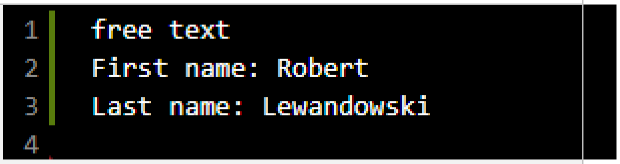
Input/Output description

The program uses a text file as input and outputs transformed text into output file. The transformation works as it was previously described. Basic example of usage:

Input	Output


`#MDEF A`
`free $0 text $1 A`
`#MEND`
`free text`
`#MCALL A X,Y`
	
	`free text`
	`free X text Y A`

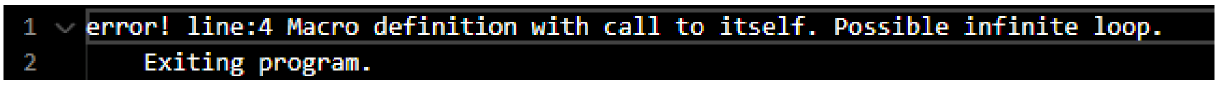
Input	Output


`#MDEF Person`
`First name: $0`
`Last name: $1`
`#MEND`
`free text`
`#MCALL Person Robert,Lewandowski`
	
	`free text`
	`First name: Robert`
	`Last name: Lewandowski`

Errors and Warnings description

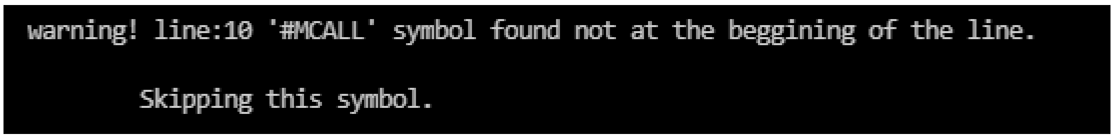
During program execution some problems can be encountered when transforming text. In case of an error, the program prints an error to console and stops execution of the program. In case of an warning program prints message to console and continues transformation.

Error syntax:



```
error! line:4 Macro definition with call to itself. Possible infinite loop.
    Exiting program.
```

Warning syntax:

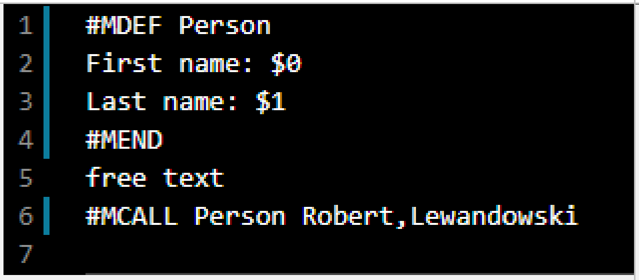


```
warning! line:10 '#MCALL' symbol found not at the beggining of the line.

        Skipping this symbol.
```

IV. Functional test cases

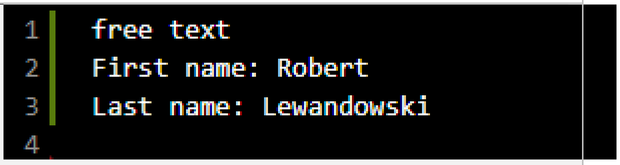
Case 1: Basic functionality

Input



```
#MDEF Person
First name: $0
Last name: $1
#MEND
free text
#MCALL Person Robert,Lewandowski

```

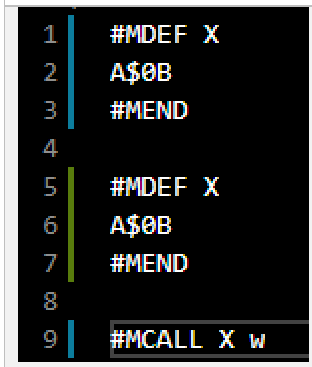
Output



```
free text
First name: Robert
Last name: Lewandowski

```

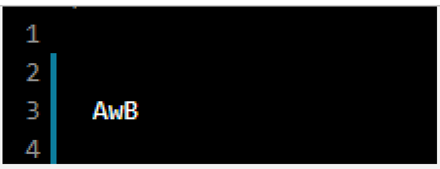
Case 2: Macro redefinition

Input



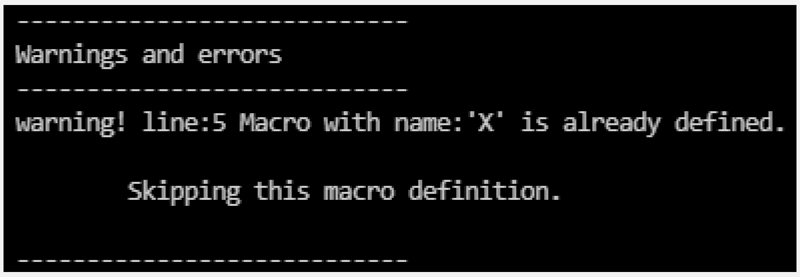
```
1  #MDEF X
2  A$0B
3  #MEND
4
5  #MDEF X
6  A$0B
7  #MEND
8
9  #MCALL X w
```

Output



```


AwB

```

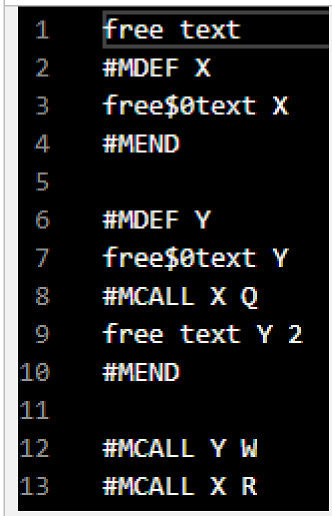
Console:



```
----------------------------
Warnings and errors
----------------------------
warning! line:5 Macro with name:'X' is already defined.

        Skipping this macro definition.

----------------------------
```

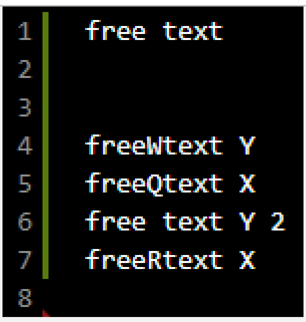
Case 3: Nested #MCALL

Input



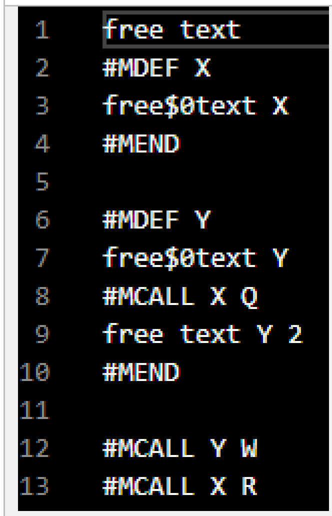
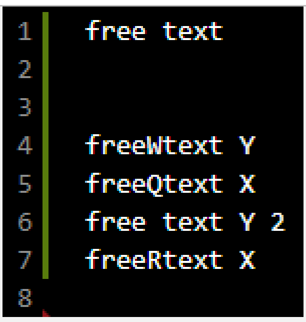
```
free text
#MDEF X
free$0text X
#MEND

#MDEF Y
free$0text Y
#MCALL X Q
free text Y 2
#MEND

#MCALL Y W
#MCALL X R
```

Output



```
free text


freeWtext Y
freeQtext X
free text Y 2
freeRtext X

```

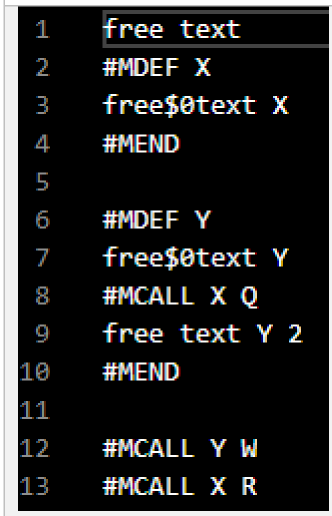
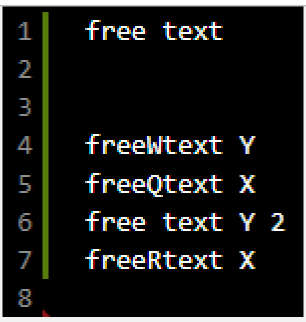
Case 4: Nested #MCALL

Input	Output
	

```
free text
#MDEF X
free$0text X
#MEND

#MDEF Y
free$0text Y
#MCALL X Q
free text Y 2
#MEND

#MCALL Y W
#MCALL X R
```

```
free text


freeWtext Y
freeQtext X
free text Y 2
freeRtext X

```

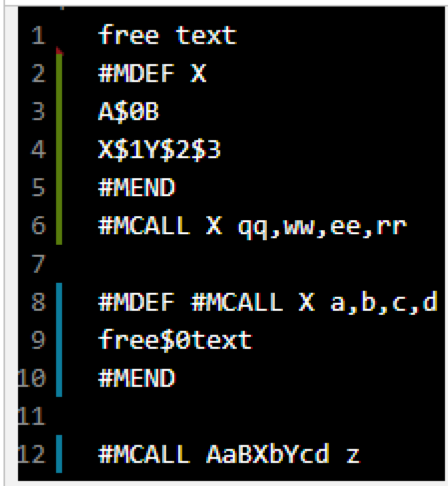
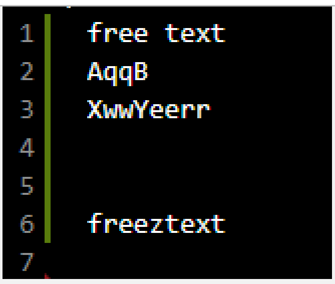
Case 5: Nested macro call

Input	Output
	

```
free text
#MDEF X
free$0text X
#MEND

#MDEF Y
free$0text Y
#MCALL X Q
free text Y 2
#MEND

#MCALL Y W
#MCALL X R
```

```
free text


freeWtext Y
freeQtext X
free text Y 2
freeRtext X

```

Case 6: Nested macro call as macro definition name

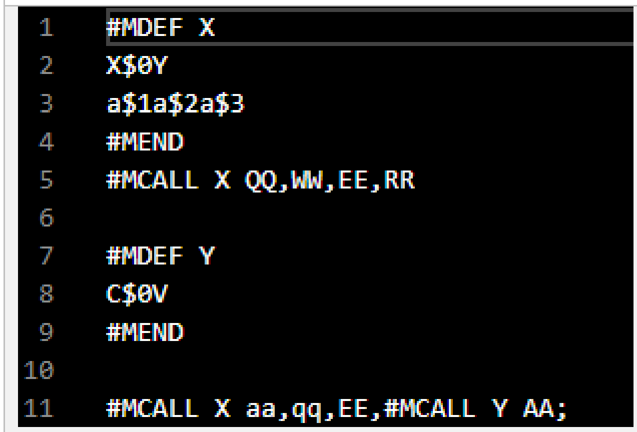
Input	Output
	

```
free text
#MDEF X
A$0B
X$1Y$2$3
#MEND
#MCALL X qq,ww,ee,rr

#MDEF #MCALL X a,b,c,d
free$0text
#MEND

#MCALL AaBXbYcd z
```

```
free text
AqqB
XwwYeerr


freeztext

```

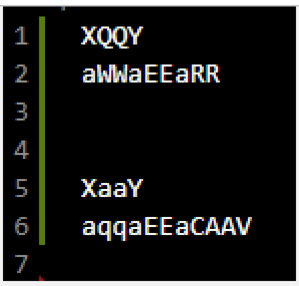
Case 7: Nested macro call passed as actual parameter

Input



```
#MDEF X
X$0Y
a$1a$2a$3
#MEND
#MCALL X QQ,WW,EE,RR

#MDEF Y
C$0V
#MEND

#MCALL X aa,qq,EE,#MCALL Y AA;
```

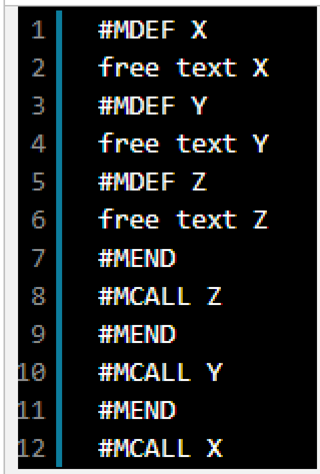
Output



```
XQQY
aWWaEEaRR


XaaY
aqqaEEaCAAV

```

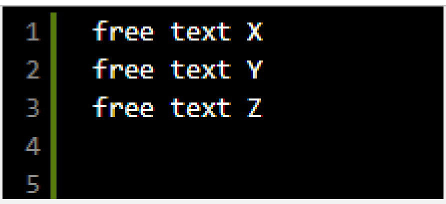
Case 8: Triple nested macro definitions

Input



```
#MDEF X
free text X
#MDEF Y
free text Y
#MDEF Z
free text Z
#MEND
#MCALL Z
#MEND
#MCALL Y
#MEND
#MCALL X
```

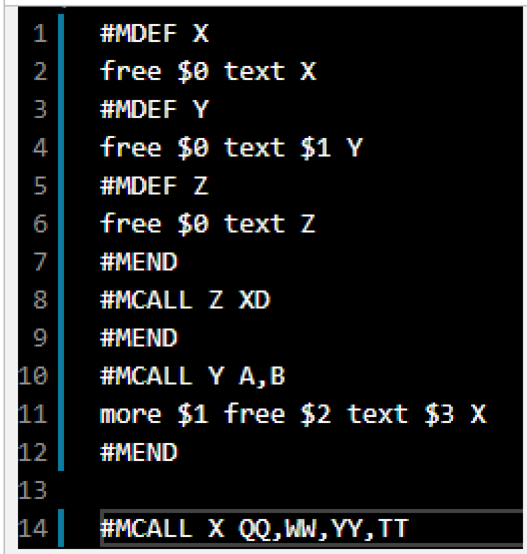
Output



```
free text X
free text Y
free text Z


```

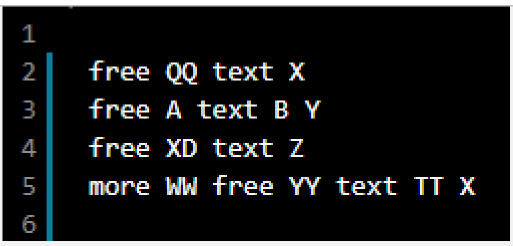
Case 9: Positional parameters in nested macro definitions

Input



```
#MDEF X
free $0 text X
#MDEF Y
free $0 text $1 Y
#MDEF Z
free $0 text Z
#MEND
#MCALL Z XD
#MEND
#MCALL Y A,B
more $1 free $2 text $3 X
#MEND

#MCALL X QQ,WW,YY,TT
```

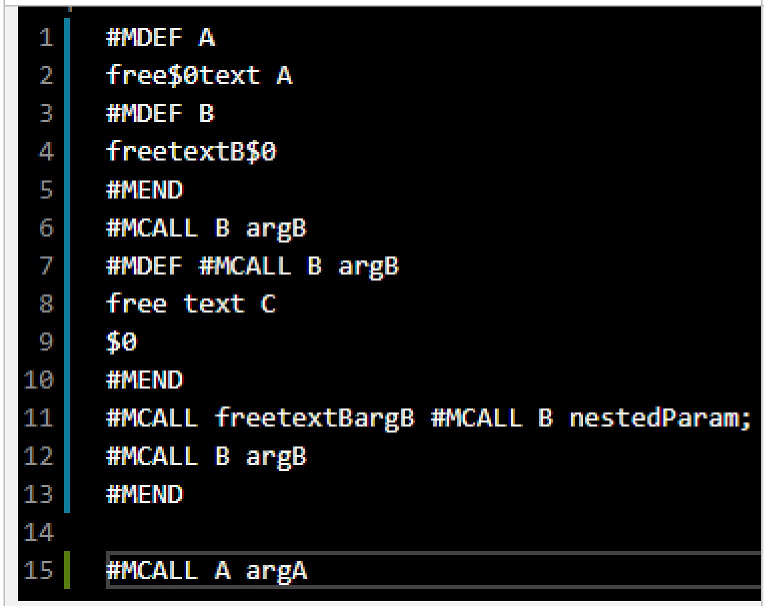
Output



```

free QQ text X
free A text B Y
free XD text Z
more WW free YY text TT X

```

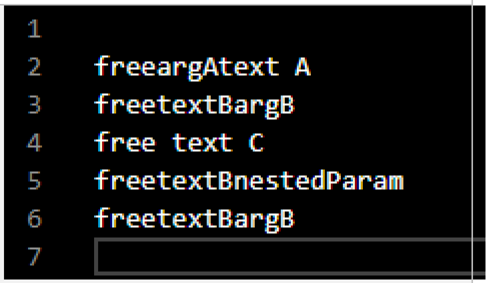
Case 10: Advanced nesting

Input



```
#MDEF A
free$0text A
#MDEF B
freetextB$0
#MEND
#MCALL B argB
#MDEF #MCALL B argB
free text C
$0
#MEND
#MCALL freetextBargB #MCALL B nestedParam;
#MCALL B argB
#MEND

#MCALL A argA
```

Output



```

freeargAtext A
freetextBargB
free text C
freetextBnestedParam
freetextBargB

```