SYSTEMS

PROGRAMMING LAB

MACRO

PREPROCESSOR

DOCUMENTATION

-BY TARISHI JAIN

ID:2016UCP1443

BATCH:B1

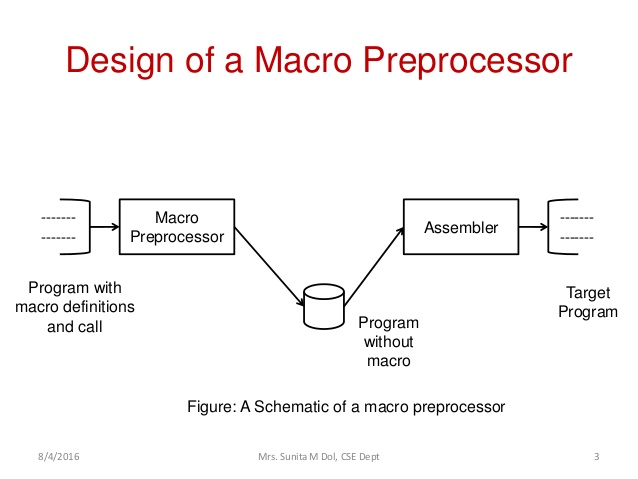
(Under guidance of Professor Arka Prokash Mazumdar)

What are macros?

* Macros are name for some fragment of code.
* A macro is a symbol that is recognised by the pre-processor and replaced by the macro body.
* So wherever the name is used it get replaced by the fragment of code by the pre-processor program.
* It consists of a name, a set of formal parameters and a body of code.
* Thus they provide a program generation facility through macro expansion.

Design of a macro pre-processor:

* The macro pre-processor accepts an assembly program containing definition and calls and translates it into an assembly program which does not contain any macro definition call.
* The design of a macro processor generally is machine independent!



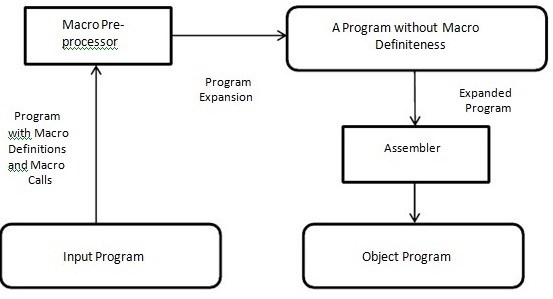
MACROS vs FUNCTIONS:

Argument Evaluation

* A function argument is evaluated once, before the call.
* A macro argument is evaluated each time it is encounters in the macro body.

Macro pre-processor

* A macro pre-processor essentially accepts an assembly program with macro definitions and calls as its input and processes it into an equivalent expanded assembly program with no macro definitions and calls.
* The macro pre-processor output program is then passed over to an assembler to generate the target object program.

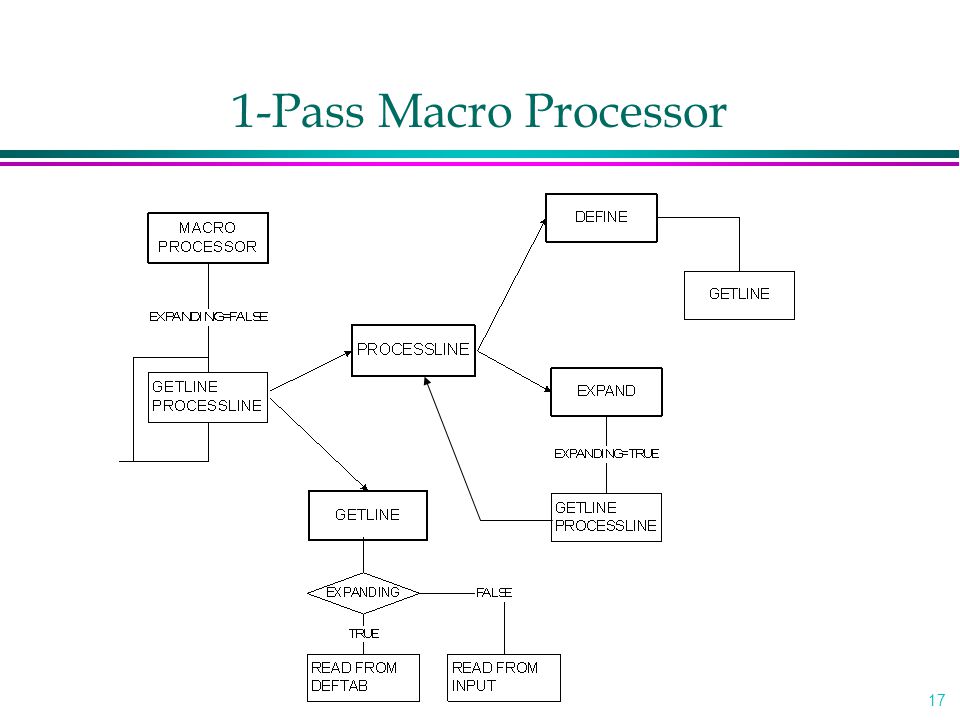


One pass macro pre-processor:

* A one pass macro pre- processor alternate between macro definition and macro expansion in a recursive way and is able to handle recursive macro definition.

Restriction:

* The definition of macro must appear in the source program before any statements that invoke that macro.
* This restriction does not create any real inconvenience.



Multi-purpose one pass MACRO Pre-Processor designed in the Project:

* It is suitable for Low-level languages like NASM/GAS.Also for high level languages like C, Python.
* Macro pre-processor code written in Python3.

Features:

* It supports Single line and multi-line comments for programmer’s convenience. The macro pre-processor removes the comments.
* Allows nested macro definitions.
* Supports single line macro definitions.
* Supports multi line macro definitions
* Supports parameter substitution(positional+default).
* Supports conditional macro definition.
* It is suitable for low level as well as high level languages.

Syntax and prototype:

Two assembler directives are used:

* MACRO: identify the beginning of macro definition.
* MEND: identify the end of macro definition.

Macro definitions:

* Single line macro:



Calling example:<macro name>()

* Multi line macro:
* With parameters:

C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture.png

Calling example:<macro name> (par1,par2)

* Using default parameters:

Calling:



Definition:



,where <value> is the default value of parameter3.

* Without parameters:



Calling example:<macro name>

* macro-name: the name of the macro.
* parameters: (these are optional)the parameters to be passed in the macro body(it may also take default values).
* definition: body of the macro, which will be replaced by the macro invocation.

Comments:

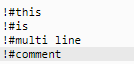
* Single line comment:

Ex:

C:\Users\user\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture.png

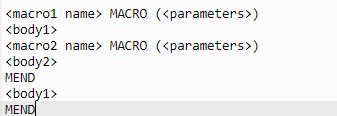
* Multi line comment:

Ex:



Nested:

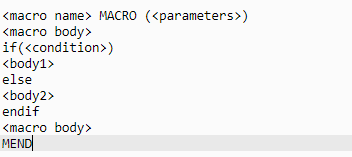
Nested macro prototype:



* <macro1 name>: name of first macro
* <macro2 name>:name of inner macro
* <body1>:body of outer macro
* <body2>:body of inner macro
* <parameters>:list of parameters

Conditional macro:

* Macro definition will be accompanied by conditions.
* Only that part of the macro definition will be expanded which holds true.



* A helper.txt file is created to store the condition expression.
* Another python file(named evaluate.py) is used to check whether that condition is true or not.
* The function in that file returns True if that condition is true, else False.

Data structures for one-pass macro processor:

DEFTAB (definition table) :

* Stores the macro definition including macro prototype and macro body .
* Comment lines are omitted.
* References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.

NAMTAB :

* Stores macro names .
* Serves as an index to DEFTAB .
* Pointers to the beginning and the end of the macro definition (DEFTAB) .

ARGTAB :

* Stores the arguments of macro invocation according to their positions in the argument list.
* As the macro is expanded, arguments from ARGTAB are substituted for the corresponding parameters in the macro body.

Implementation:

* The input file is read line by line.
* If “MACRO” is found, then it implies macro definition.the name of the macro is taken from the line and written in nametab.txt file,which will contain the name of all the macros in the code.
* A loop will run from that line till keyword “MEND” is encountered ,which indicates the end of macro definition .In this loop,the definition of the macro will be stored in deftab.txt,for further use by macro pre-processor.
* Both the types of comments are deleted by the macro pre-processor by skipping commented lines.
* If inside a macro definition,”MACRO” keyword appears,then it implies nested macro:
* The name of the inner macro will be stored in namtab.txt.Its definition in deftab.txt and arguments in argtab.txt.
* The syntax used to store definition in deftab.txt is:

(example for 2 macro definition):

<macro1><arguments>

<definition>

…

<macro2><arguments>

<definition>

…

The three dots(…) are used to indicate end of macro definition.

* After macro definition ,main code starts with the following syntax:

Main body:

<main code goes here>

* If macro name is found in main code, then the parameters are written in argtab.txt, then the required macro definition is searched in deftab.txt, followed by its expansion in the main code with the arguments passed.
* The process continues until EOF is not reached.

How to run(Ubuntu) ?

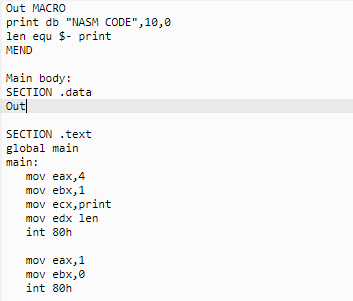
* Go to the terminal and type the following commands:
* chmod a+x file.py
* python3 file.py input.txt

,where input.txt is the input file containing the main code.

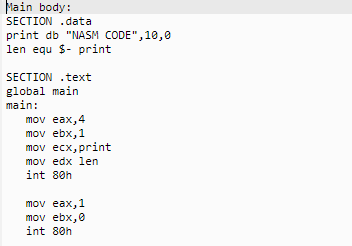
Some Examples:

Multi line macro (without arguments):

* Ex.Input.txt: (NASM)



Output generated:



Following three tables were created by the pre processor:

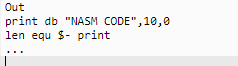
* Nametab.txt



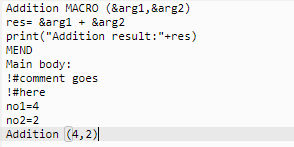
* Argtab.txt

Empty (since no parameter)

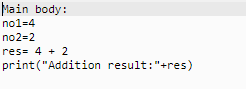
* Deftab.txt



* Ex.Input.txt: (Python)



Output generated:



Comment is no more present in output file.

* Nametab.txt:



* Argtab.txt:

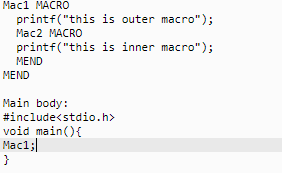


* Deftab.txt: ?1 depicts argument1,?2 depicts argument2 and so on..



* Ex. Input.txt: (C)

Nested macro:



Output generated:

