Yeditepe University
Department of Computer Engineering

CSE 232
Systems Programming
Spring 2017

Term Project

Macroprocessor

Due to: 7th May 2017

3 Students in a Group

Write a macroprocessor to expand macros given in the following format. Your macroprocessor will work as a preprocessor to M6800 assembler.

Write your macroprocessor in C and use gcc compiler on Linux.

@DEFINE construct:

@DEFINE *name* *replacement-text*

The *name* and *replacement-text* must be saved in a table. All subsequent occurrences of the *name* must be replaced by the *replacement-text*. (Ex: @DEFINE K 3)

@MACRO - @ENDM construct:

@MACRO *macro-name* (*param1, param2, param3)*
… (macro body)
@ENDM

This is a macro definition. There may be maximum 3 parameters in the parameter list of a macro (*param1, param2, param3*). The *macro-name, parameter list* and the code between MACRO and ENDM (macro body) must be saved in a buffer. All subsequent occurrences of the *macro-name* must be replaced by the macro body, substituting the actual parameters in the place of dummy parameters.

@MCALL construct:

@MCALL *macro-name* (*rparam1, rparam2, rparam3*)

This is a macro call and it must be expanded by the macroprocessor, substituting the real parameters (*rparam1*, *rparam2, rparam3*) in the places of dummy parameters.

@IF construct:

```
@IF (condition)
  …
@ELSE
  …
@ENDIF
```

After evaluating the *condition*, if the result is true, the code between @IF and @ELSE must be included in the code, otherwise the code between @ELSE and @ENDIF must be included in the expanded code.

condition is defined as:

<condition> ::= <name> <op> <value>

where <op> is '<' or '=' or '>', and <name> is a name defined in a previous @DEFINE construct, and <value> is the replacement-text in that definition.

Macroprocessor

At the beginning, define the tables using the following data structures:

- *Def_symbols* table: contains the names and the replacement-texts defined by @DEFINE

```
struct symbol {
      char name[10];     // symbol name
      char value[10];    // replacement text
}
struct symbol def_symbols[50];
```

- *Macros* table: contains the names, parameters and codes of the macros

```
struct mac {
      char mname[10];   // macro name
      int nparams;      // number of parameters
      char p[3][10];    // dummy parameters
      char mbody[1000]; // macro body
}
struct mac macros[20];  // maximum 20 macros can be defined
```

STEP 1:

Read the defined symbols and the macro definitions at the beginning of a given input file and fill the *Def_symbols* and *Macros* tables.

@DEFINE
Insert the *name* and *replacement-text* in the *Def_symbols* table as character strings.

@MACRO - @ENDM
Insert the *macro-name*, parameters and the code in the macro body in the *Macros* table as character strings.

Then, print the contents of the tables.

STEP 2:

Read the rest of the input file line by line. If the line does not contain a macro construct, write it to the output file. If it contains a macro construct, expand it and write the expanded code to the output file.

@MCALL
Search *macro-name* in the *Macros* table. If it is not found, print an error message. If found, create a *Parameters* table and insert the dummy and real parameters in this table. Then, take the macro body from *Macros* table and write it to the output file, substituting the parameters. Check each line in the macro body. If it doesn't contain a parameter, no substitution is necessary, write it in the output file. If it contains a parameter, write it in the output file, substituting the real parameters in the dummy parameters.

Parameters table must have the following data structure:

```
struct param {
      char dp[3][10];   // dummy paramaters
      char rp[3][10];   // real paramaters
}
struct param PT[5];     //maximum 5 nested macro calls
```

@IF construct:
In order to check the *condition*, search the *name* in *Def_symbols* table and compare its *value* with the *replacement-text*. If the condition is true, then copy the code between IF-ELSE to the output file, otherwise copy the code between ELSE-ENDIF to the output file. The code between IF-ELSE or ELSE-ENDIF may contain a macro call. In this case, your macroprocessor must also expand the macro with parameter substitution.

Example:

You may test your macroprocessor with the following code. The expanded code must be written to a file and the tables must be printed on the screen. Change the value of N1 and test both cases.

Source Code	Expanded Code

`@DEFINE N1 5`
 	
`@MACRO MOVE (A, B)`
`LDAA A, X`
`INCA`
`STAA B, X`
`@ENDM`
 	
`RORG`
`LDX 100H`
`LDAB #10H`
`@MCALL MOVE (30H, 40H)`
`LOOP: CMPB #0H`
`BEQ DONE`
`@IF (N1=5)`
`@MCALL MOVE (0H, 20H)`
`@ELSE`
`@MCALL MOVE (20H, 0H)`
`@ENDIF`
`INX`
`DECB`
`BRA LOOP`
`END`
	`RORG`
	`LDX 100H`
	`LDAB #10H`
	`LDAA 30H, X`
	`INCA`
	`STAA 40H, X`
	`LOOP: CMPB #0H`
	`BEQ DONE`
	`LDAA 0H, X`
	`INCA`
	`STAA 20H, X`
	`INX`
	`DECB`
	`BRA LOOP`
	`END`

BONUS (20 points)
If your macroprocessor can expand recursive macro calls (macro calls within macros), you may get 20 points bonus.