

CSI 333 – Programming at the Hardware/Software Interface – Fall 2011

Programming Assignment IV

Date given: Oct. 25, 2011

Due date: Nov. 11, 2011

Weightage: 10%

This is a *team* project (except for those students who have opted to work on their own). The regular deadline for this assignment is **11 PM, Friday, November 11, 2011**. With lateness penalty, the program will be accepted until **11 PM, Sunday, November 13, 2011**. The assignment will *not* be accepted after that deadline.

Important notes: The C source file for the program must be named `p4.c`. This file must be submitted using the `turnin-csi333` command by the deadline specified above. *Each team must make only one submission*. Additional information about the `turnin-csi333` command will be included in the README file for this assignment.

You must follow the programming and documentation guidelines indicated in the handout on “Course Policies”. Your program must have several functions in addition to `main`. Some suggestions regarding this will be provided in class.

You are required to write a C program whose input is a MIPS Assembly Language (MAL) program and whose output is a listing of the MAL program with line numbers and/or a **cross reference table** for the MAL program. Details regarding MAL programs and the contents of cross reference tables are given below.

Command Line Details: Suppose the executable version of your program is named `prog4.out`. The program will be executed by a command line of the following form:

```
prog4.out flag inputfile outputfile
```

In the command line above, the arguments *inputfile* and *outputfile* specify the names of the input file (which contains a MAL program) and the output file respectively. A valid *flag* argument is one of `-l`, `-c` or `-b`. If the *flag* is `-l`, your program must produce *only* a listing of the MAL source program in the specified output file. If the *flag* is `-c`, your program must produce *only* the cross reference table for the MAL source program in the specified output file. If the *flag* is `-b`, your program must produce *both* the listing and the cross reference table in the specified output file.

Details Regarding MAL Programs: Each line of a MAL program may be a blank line (containing only white-space characters), a comment line (a line that starts with the character `'#'`) or a MAL statement. Each MAL statement has an optional label field (terminated by a colon), a mandatory opcode field, an optional operand field consisting of zero or more operands separated by commas, and an optional comment field (starting with the character `'#'`). The different fields are separated by one or more spaces or tabs.

An identifier in a MAL program is *defined* in the source line where the identifier appears as a *label* and is *used* in lines where it appears in the *operand* field. (Note that opcodes are *not* identifiers. Also, operands starting with the symbol `$` are *not* identifiers; they represent registers of the MIPS machine.)

Example: Consider the following MAL statement:

```
iloop_beg: la $7,arr      #Get address of arr into register 7.
```

In the above MAL statement, the label field has the identifier `iloop_beg`. The opcode field has `la`. The two operands are `$7` (a register – *not* an identifier) and `arr` (an identifier). The operand field is followed by the comment field that starts with the `#` character. The above MAL statement defines the identifier `iloop_beg` and uses the identifier `arr`.

A **cross reference table** for a MAL program indicates for each identifier, the line number in the source program where the identifier is defined and the line number(s) in the source program where the identifier is used. An example of a MAL program and its cross reference table are shown on pages 3 and 4 of this handout.

Here are some additional details regarding MAL programs.

1. Every line of a MAL program is terminated by the newline (`'\n'`) character. There are at most 80 characters (including the newline character) on each line.
2. Any line that has `'#'` as the first character is a comment line.
3. In any non-comment non-blank line, the label, opcode, operand and comment fields are separated by one or more spaces or tabs. If such a line has a label, the label will start from the first position. (Thus, a line that starts with a space or tab does not have a label.)
4. The first character of any identifier is a lower or upper case letter or the underscore character; this is followed by zero or more upper/lower case letters, digits or underscore characters. The identifiers are *case sensitive*. (Thus, `Val` and `val` represent different identifiers.) The maximum length of an identifier is 10. The maximum number of labels that can appear in any MAL program is 100.
5. If the operand field of a MAL program starts with the single quote character (`'`) or the double quote character (`"`), then the operand field does *not* have any identifiers. (The reason is that an operand field starting with the single quote specifies a literal character; an operand field starting with the double quote character specifies a string.)

Be sure to study the example given on pages 3 and 4 of this handout before writing your program.

Errors to be detected: You need to check only for the usual command line errors (wrong number of parameters on the command line, invalid flag, the input or the output file can't be opened). In such a case, your program must output a suitable error message to `stderr` and stop.

Information About the README file: By 10 PM on Tuesday, November 2, 2011, the directory `~csi333/public/prog4` on the ITS Unix machines will contain a file called `prog4.README`. This file will have the information on how you can turn in your source file electronically. The file may also contain information about some sample input and output files that you can use to test your program.

Example: Suppose the file `example.mal` contains the following MAL program.

```
#A sample MAL program.

        .data                #Data segment begins here.
avg:    .word                #Will store the average.
i1:     .word                20    #First integer.
i2:     .word                13    #Second integer.
i3:     .word                82    #Third integer.
prompt: .asciiz             "Value is: "
nl:     .byte                '\n'

        .text                #Text segment begins here.
__start: lw                 $15,i1    # $15 contains 20.
        lw                 $16,i2    # $16 contains 13.
i10:    add                 $15,$15,$16 #Operand field has no ids.
        lw                 $16,i3    # $16 contains 82.
        add                 $15,$15,$16 # $15 contains the sum (115).
        li                 $16,3     # $16 contains 3.
        div                 $15,$15,$16 # $15 contains the average (38).
i20:    sw                 $15,avg    #Store the average.
        puts                prompt
        put                 avg
        putc                nl

        sw                 $15,avg
        la                 $16,i1
        sw                 $15,0($16)
        add                 i3,i3,1
        done                #Similar to halt.
```

After we execute your program using the command line

```
prog4 -b example.mal output.lst
```

the contents of the output file `output.lst` should be as shown on page 4 of this handout. In studying the output file, you should keep the following in mind.

1. The line numbers start at 1.
2. Blank lines in the input file are echoed to the output file; however, they are *not* numbered.
3. For each identifier, the line numbers where it is used are in *increasing order*.
4. When an identifier is used several times in a source line (e.g. the identifier `i3` appears twice in line 24), the line number appears only once in the list.

Contents of the file output.lst:

```
1  #A sample MAL program.
2      .data                #Data segment begins here.
3  avg:    .word            #Will store the average.
4  i1:     .word            20    #First integer.
5  i2:     .word            13    #Second integer.
6  i3:     .word            82    #Third integer.
7  prompt: .asciiz         "Value is: "
8  nl:     .byte            '\n'

9      .text                #Text segment begins here.
10 __start: lw             $15,i1    # $15 contains 20.
11         lw             $16,i2    # $16 contains 13.
12 i10:    add            $15,$15,$16 #0operand field has no ids.
13         lw             $16,i3    # $16 contains 82.
14         add            $15,$15,$16 # $15 contains the sum (115).
15         li             $16,3     # $16 contains 3.
16         div            $15,$15,$16 # $15 contains the average (38).
17 i20:    sw             $15,avg    #Store the average.
18         puts          prompt
19         put           avg
20         putc          nl

21         sw             $15,avg
22         la             $16,i1
23         sw             $15,0($16)
24         add            i3,i3,1
25         done           #Similar to halt.
```

Cross Reference Table

Identifier	Definition	Use
avg	3	17 19 21
i1	4	10 22
i2	5	11
i3	6	13 24
prompt	7	18
nl	8	20
__start	10	
i10	12	
i20	17	