CSI 333 – Programming at the Hardware-Software Interface  
SQUPT, Spring 2019

**Project 4**

The total grade for the assignment is 100 points.

You must follow the programming and documentation guidelines (see file *Programming Assignments Requirements and Recommendations.docx*).

This is a **team project** (except for those students who have opted to work on their own). Group of **two** students may work on this project together.

**Due date: 11:59pm Sunday, May 5, 2019**

# Description

You are required to write a C program

* whose input is a MIPS Assembly Language (MAL) program and
* whose output is a list of labels of the MAL program used as variables, a list of labels of the MAL program used for the flow control, or both.

Details regarding MAL programs and lists of labels are given below.

Command Line Details: Suppose the executable version of your program is named p4.exe (if Windows). The program will be executed by a command line of the following form:

p4.exe 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 -v, -f or -b.

* If the flag is -v, your program must produce only a list of labels of the MAL source program used as variables in the specified output file. (Variable labels are identifiers defined in the .data segment of a MAL program.)
* If the flag is -f, your program must produce only a list of labels of the MAL program used for the flow control in the specified output file. (Flow control labels are identifiers defined in the .text segment of a MAL program.)
* If the flag is -b, your program must produce (i) a list of labels of the MAL source program defined as variables, and (ii) a list of labels of the MAL program defined for flow control in the specified output file.

In some MAL programs, source code may contain more than one .data and .text parts (for example, in case of procedure calls each procedure may have its own .data and .text parts). Your program is to work with MAL files that have only one .data and one .text parts.

Details Regarding MAL Programs:

1. Each line of a MAL program may be
   1. a blank line (containing only white-space characters),
   2. a comment line (a line that starts with the character ’#’) or
   3. a MAL statement which has the following fields separated by one or more spaces or tabs:
      1. an optional label field terminated by ’:’,
      2. a mandatory opcode field,
      3. an optional operand field consisting of zero or more operands separated by commas, and
      4. an optional comment field starting with ’#’.
2. Every line of a MAL program is terminated by the newline (’\n’) character and has at most 80 characters (including the newline character).
3. Any line that has ’#’ as the first character is a comment line.
4. The labels start from the first position, thus, a line that starts with a space or tab does not have a label.) The maximum number of labels that can appear in any MAL program is 100.
5. The first character of any identifier is a lower- or upper-case letter or the underscore character 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.
6. Opcodes are not identifiers. Also, operands starting with the symbol ‘$’ are not identifiers; they represent registers of the MIPS machine.
7. 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.)
8. An identifier in a MAL program:
   1. is defined in the source line where the identifier appears as a label and
   2. is used in lines where it appears in the operand field.

Examples:

iloop\_beg: la $7,arr #Get address of arr into register 7.

In the above, 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 statement defines the identifier iloop\_beg and uses the identifier arr.

loop: sw $15,avg #Store reg. 15

In the above instruction, the label is loop, the opcode is sw and the operands are $15 and avg. The instruction defines the identifier loop and uses the identifier avg.

An example of a MAL program and lists of labels are shown below. Be sure to study the example given below before writing your program.

Suppose the file example.asm 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 i1 = 20.

lw $16,i2 #$16 contains i2 = 13.

i10: add $15,$15,$16 #Operand field has no ids.

lw $16,i3 #$16 contains i3 = 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 the execution of the program using the command line

p4.exe -b example.asm output.txt

the contents of the output file output.txt should be as shown below. In studying the output file, you should keep the following in mind:

1. Name of the identifier in the list is surrounded with ‘-‘.
2. The identifiers in the list are in order of their definition in the program.
3. For each identifier, the source lines where it is used are in order of their appearance in the program.
4. If an identifier is not used, there is neither source line, no empty line after the identifier.
5. When an identifier is used several times in a source line (e.g. the identifier i3 appears twice in one line), the source line appears only once in the list.
6. If an identifier appears in a comment line, it is ignored it and not included it in the output file with any options.

Contents of the file output.txt:

Variable ID –avg-

i20: sw $15,avg #Store the average.

put avg

sw $15,avg

Variable ID -i1-

\_\_start: lw $15,i1 #$15 contains 20.

la $16,i1

Variable ID -i2-

lw $16,i2 #$16 contains 13.

Variable ID -i3-

lw $16,i3 #$16 contains 82. 6 add i3,i3,1

Variable ID –prompt-

puts prompt

Variable ID –nl-

putc nl

Flow Control ID -\_\_start-

Flow Control ID -i10-

Flow Control ID -i20-

Programming Suggestions:

1. Data structure to be used: array (of size 100) of struct, where each struct has the following data members:
   1. a char array of size 11 (to store an identifier),
   2. a pointer to a linked list; each node of the list stores a source line where the identifier is used.
2. Don’t assume any limit on the number of lines in the input file. Read the input line by line (using fgets). For each line of the input file which is not a comment line or a blank line, use strtok to parse the line (i.e., extract the various fields of the instruction).
3. All input files will satisfy the following conditions.
   1. The MAL program in the input file won’t contain any errors.
   2. Every identifier will be defined somewhere in the MAL program. (However, it is possible that an identifier is not used anywhere.)
   3. There will be no multiply defined identifiers.
4. 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.
5. The input and output file names must NOT be hard-coded into your program. These files names are available to your program as command line arguments.
6. Program must contain several functions in addition to main.
7. It is advisable for each team member to take the responsibility for one of the program’s major components:
   1. Processing each MAL statement (e.g. checking if the statement is a comment, a blank line or a MAL statement, identifying the label and/or the identifiers appearing in a MAL statement).
   2. Maintaining the data structure that stores the lists information.

# Example of program execution and sample data to test your program

The following examples assume that the executable version of the program is in the file p4.exe (if Windows).

Important Note: Some sample inputs that can be used to test your programs are given below. However, you should remember that when we compile and run your source files, we will use other data. Just because your programs work for the sample inputs given below, you shouldn't assume that they will work for all inputs. Therefore, you should test your programs thoroughly with other input values.

The content of input file mal\_prog.asm:

.data

x1: .word 0

x2: .word 80

.text

begin: lw $15,x1

add $15,$15,x2

beqz $15, end

sw $15,x1

end: done

Suppose we execute the following command:

> p4.exe -b mal\_prog.asm output.txt

The contents of the file output.txt:

Variable ID -x1-

begin: lw $15,x1

sw $15,x1

Variable ID –x2-

add $15,$15,x2

Flow Control ID –begin-

Flow Control ID –end-

beqz $15, end

You may use information below to create your test file and the corresponding output file. **It is not going to be the test file we will use to grade your project!**

Example to help you to test your code: Assume your project name is p4.exe and the MAL input file name is ex2.asm. Contents of input file ex2.asm:

#Example 2.  
#Prompts the user for one character. Checks whether the  
#character is a digit and prints an appropriate message.

          .data  
inchar:   .space    3  
prompt:   .asciiz  "Type a char: "  
yes\_str:  .asciiz  "Yes -- It is a digit.\n"  
no\_str:   .asciiz  "No -- It is not a digit.\n"  
  
          .text  
          .globl   main  
  
#Prompt user for character.  
main:     li       $v0, 4        #Command for print string.  
          la       $a0, prompt   #Start addr. of prompt string.  
          syscall  
  
#Read the character. (We must also read the newline character and  
#provide space for the '\0' character. So, the buffer  
#must be at least 3 characters long.)  
  
          li       $v0, 8       #Command for read string.  
          la       $a0, inchar  #Buffer  
          li       $a1, 3       #Max. no. of characters (incl. newline)  
                                #to be read = 3 - 1 = 2.  
          syscall  
  
#Check whether the character is a digit.  
  
          lbu      $4, inchar   #Get the character into $4.  
          li       $5, 48       # 48 is the ASCII code for '0'  
          li       $6, 57       # 57 is the ASCII code for '9'  
  
          blt      $4, $5, no\_part  #If ascii code is < 48 or > 57  
          bgt      $4, $6, no\_part  #it can't be a digit.  
  
#Code to print "yes" answer and stop.  
  
yes\_part: li       $v0, 4  
          la       $a0, yes\_str  
          syscall  
          li       $v0, 10  
          syscall  
  
#Code to print "no" answer and stop.  
  
no\_part:  li       $v0, 4  
          la       $a0, no\_str  
          syscall  
          li       $v0, 10  
          syscall

Running p4.exe with option -v and the output file name is out2-var.txt:

> p4.exe -v ex2.asm out2-var.txt

Contents of out2-var.txt:

Variable ID -inchar-  
la       $a0, inchar  #Buffer  
lbu      $4, inchar   #Get the character into $4.  
  
Variable ID -prompt-  
la       $a0, prompt   #Start addr. of prompt string.  
  
Variable ID -yes\_str-  
la       $a0, yes\_str  
  
Variable ID -no\_str-  
la       $a0, no\_str

Running p4.exe with option -f and the output file name is out2-flow.txt:

> p4.exe -f ex2.asm out2-flow.txt

Contents of out2-flow.txt:

Flow Control ID -main-  
Flow Control ID -yes\_part-  
Flow Control ID -no\_part-  
blt      $4, $5, no\_part  #If ascii code is < 48 or > 57  
bgt      $4, $6, no\_part  #it can't be a digit.

Running p4.exe with option -b and the output file name is out2-both.txt:  
  
> p4.exe -b ex2.asm out2-both.txt  
  
Contents of out2-both.txt:

Variable ID -inchar-  
la       $a0, inchar  #Buffer  
lbu      $4, inchar   #Get the character into $4.  
  
Variable ID -prompt-  
la       $a0, prompt   #Start addr. of prompt string.  
  
Variable ID -yes\_str-  
la       $a0, yes\_str  
  
Variable ID -no\_str-  
la       $a0, no\_str  
  
Flow Control ID -main-  
Flow Control ID -yes\_part-  
Flow Control ID -no\_part-  
blt      $4, $5, no\_part  #If ascii code is < 48 or > 57  
bgt      $4, $6, no\_part  #it can't be a digit.

# Submission

You must perform submissions as directed by your co-instructor.

*Ignoring any of the following rules will result in penalty or even ZERO grade for the project!*

For the **team project** each team must make **only one** submission. That is, in each team, ONLY ONE member must do this. Team submissions must include additional documentation in the source file as explained below.

Submission should include:

* A file named as directed by your co-instructor with source code for the project. More details will be given in your lab classes.
* Screenshot with program output.

At the top of your source code file the following information must appear in the form of comments:

* 1. Course code and title (i.e. “CSI 333. Programming at the Hardware-Software Interface”),
  2. Semester (e.g., Spring 2019),
  3. The name of your lab classes supervisor,
  4. Your class (e.g., ZR170102),
  5. Your student ID,
  6. Your pinyin name.

Students working in a team must have the following information at the beginning of your source file in the form of comments:

* 1. Course code and title (i.e. “CSI 333. Programming at the Hardware-Software Interface”),
  2. Semester (e.g., Spring 2019),
  3. The name of your lab classes supervisor,
  4. Your class (e.g., ZR170102),
  5. The student IDs of the two team members,
  6. The pinyin names of the two team members,
  7. A clear explanation of how the work for the project was divided among the two team members. Indicate clearly who developed each function and how the testing work was divided between the team members.

Make sure that your program compiles and produces correct results on the lab machines. Programs that cause compiler or linker errors on these machines will NOT receive any credit.

# Project Grading

Programs will be graded by co- instructors.

For students working individually:

1. Correctness: 85 points
2. Structure and documentation: 15 points

For students working in a team:

1. Correctness: 65 points
2. Structure and documentation: 15 points
3. Team work: 20 points

Each team member must participate in developing, documenting and testing the program. Each team should include additional documentation at the beginning of the source file indicating how the work for the project was divided between the two team members. (Indicate clearly who developed each function and how the testing work was divided between the team members.) After the submission deadline, each team must meet with their instructor who supervises the lab classes. During the meeting, the instructor will ask questions about the team’s program and determine the points for team work. (The two team members may receive different scores for team work.)

# Additional comments

Some students may use strtok library function to tokenize lines read from input file. However, you may need to use nested tokenization which will create a problem in the second call with NULL pointer. Consider this example from the web:

#include <string.h>

#include <stdio.h>

int main(void){

char str[] = "a;b;c;d;e\nf;g;h;i;j\n1;2;3;4;5\n";

char \*token = strtok(str, "\n");  
  
while(token != NULL) {

printf("a = %s\n", token);

char \*token2 = strtok(token, ";");

while(token2 != NULL) {

printf("b = %s\n", token2);

token2 = strtok(NULL, ";");

}

token = strtok(NULL, "\n");

}

return 0;

}

when you run this code, you will get this:

a = a;b;c;d;e

b = a

b = b

b = c

b = d

b = e

as you can see, strtok splits only the first line and the rest are omitted. Instead of strtok use strtok\_r. The same above example but with strtok\_r:

#include <string.h>

#include <stdio.h>

int main(void){

char str[] = "a;b;c;d;e\nf;g;h;i;j\n1;2;3;4;5\n";

char \*end\_str;

char \*token = strtok\_r(str, "\n", &end\_str);

while (token != NULL) {

char \*end\_token;

printf("a = %s\n", token);

char \*token2 = strtok\_r(token, ";", &end\_token);

while (token2 != NULL) {

printf("b = %s\n", token2);

token2 = strtok\_r(NULL, ";", &end\_token);

}

token = strtok\_r(NULL, "\n", &end\_str);

}  
return 0;

}

and the output is:

a = a;b;c;d;e

b = a

b = b

b = c

b = d

b = e

a = f;g;h;i;j

b = f

b = g

b = h

b = i

b = j

a = 1;2;3;4;5

b = 1

b = 2

b = 3

b = 4

b = 5

Good luck!