# Lab 6

# Control Structures

In this lab you will be introduced to basic control structures through the development of a small utility. Covered topics include IF, LOOPs.

Part 1

Most if not all high level languages will offer I/O to read and write integers. For the most part MIPS does as well, at least to standard I/O (console). File i/o is a different story, all file i/o is characters. If a number appears in a text file, it is represented as a series of characters and not a number. E.g. if 554 were to appear in a file, it would be a string and not a number. After reading the string into a buffer we would then have to run “554” through an algorithm to produce the integer 554. Most languages have a utility to perform this action, commonly called aToi (which is a C function). Let us code such a utility.

If we read in a character it is ascii (which is nothing but a number), so subtracting char ‘0’ from this would result in an integer value representing the digit. Suppose **AnInt=0** represents the output of the conversion. To convert a string of digits we AnInt\*10 + new\_digit we just converted. The below code will perform this operation, for now our input integer string is hard coded.

Once you type in the code, be sure to print out the converted integer and exit the program gracefully. The converted number will be in $t6.

Part 2

Usually numbers represented as strings are not cleanly represented, for example there may be leading or trailing blanks. The conversion can be represented as follows:

**while the char is not in the range ‘0’..’9’ skip over it
while we have char ‘0’..’9’ convert**

****The first part can be read as follows, if char > ‘9’ or char < ‘0’ loop. How will this look. Integrate this code into the above. Notice “nine” is an ascii character defined like the zero. Once complete test your code by putting other non-digits in front of the test string. Note, if the two conditional tests both fail we have found a digit. These are the if statements.

Test this by adding some blanks and non-digits in front of the number in the string **someNum**:

Part 3

Now, what happens if we have blanks or non-digits after the number. We want to break out of the convert loop if we detect a non-digit. We have already written the code for this, the beqz $t1, exit (which detects a null char), can be replaced with a more specific test. Hint: its 2 lines of code. Test your code by adding blanks and other chars after the number in the string **someNum**.

Part 4

Consider 1 more small addition, modify the code so a string with multiple integers can be scanned. That is, can you find and extract all integers in the input string. After you have printed the integer, reset the register $t6 (used to convert the number) to 0 and branch back to where you start *skipLead*. The syscall on line 47 will print a blank between the integers. A blank is character 32. Test this addition by putting multiple integers within the string.

Part 5

If you notice, the output also printed the 0 and the 9 from the character definition section. There was nothing stopping the $t2 pointer from running past the end of someNum and into the memory defined after in the data section of the program. We removed that condition in Part 3 ( it was in the wrong place). You may have noticed, the program actually crashed. Let’s fix this. At the end of an ascziiz string is a null character, which is char 0. Add an extra if statement to exit the loop if a char 0 is detected. Char 0 is different then the char ‘0’. It is the first character in the ascii alphabet.

Think about what the algorithm needs to do:

skip all non digits, and exit program if char 0
while we have digits
 convert the digits to a number

start over

It makes sense the exit goes into the skip portion, since we technically can have an input with no digits. Modify your code to make it work.

Part 6

Clean up your code and add proper comments, so you know what is going on. This code will likely be used in upcoming labs and possibly an assignment.