Lab 4: Syntax Checker

Due Thursday 25 February 2021, 11:59 PM

Minimum Submission Requirements

- Ensure that your Lab4 folder contains the following files (note the capitalization convention):
  - Lab4.asm
  - README.txt
  - test1.txt
  - test2.txt
  - test3.txt
- Commit and push your repository
- Complete the Google Form with the correct commit ID of your final submission

Lab Objective

In this lab, you will develop a simple syntax checker that opens a file and determines whether it has balanced braces, brackets, and parentheses. This type of syntax checking is often used in programs to determine the point of a “syntax error” that the programmer needs to fix. You will use MIPS and the stack to check the balance and report either the location of a mismatch or the number of matched items on success.

Lab Preparation

1. Read sections 6.4, 8.2, 9.2 from Introduction To MIPS Assembly Language Programming.

Specification

You will write a program in the MIPS32 language using the MARS integrated development environment to check the balance of braces {}, brackets [], parentheses (). On an error, you will report a 32-bit integer byte offset (starting from 0) and which brace, bracket or parenthesis is unmatched. On success, you will print the number of matched braces, brackets and parentheses (collectively referred to as “braces”).

Input

You will be using program arguments instead of a syscall to take user inputs. See this document on how to use program arguments. The program argument will specify a file that is to be read through and checked for syntax errors. The file will contain printable ASCII characters only.

Sample Outputs

Working input file TEST.txt containing: ()[{(hello)}]

```
You entered the file:
TEST.txt

SUCCESS: There are 4 pairs of braces.

-- program is finished running --
```

*Note that the above uses “braces” to collectively refer to all of the braces, brackets and parentheses.

Improper filename error due to first character not being a letter:

```
You entered the file:
123_TEST.txt

ERROR: Invalid program argument.

-- program is finished running --
```

Improper filename error due to being too long:

```
You entered the file:
TEST_123456789012345.txt

ERROR: Invalid program argument.

-- program is finished running --
```

Proper file with brace mismatch for TEST.txt containing: [a[b[c[d]c]b}a]

```
You entered the file:
TEST.txt

ERROR - There is a brace mismatch: } at index 12

-- program is finished running --
```

*Note that above the first char “[” is at index/byte offset 0, “a” is at index/byte offset 1 and so on to get “}” at index 12. In this example, the closing brace doesn't match the brace on the stack so you report the ending mismatched index and brace.

Proper file with brace mismatch for TEST.txt containing: [a[a]a]a]

```
You entered the file:
TEST.txt

ERROR - There is a brace mismatch: ] at index 8

-- program is finished running --
```

In this example, there is an extra closing brace and the stack is empty so you just report the extra brace.

Proper file with multiple braces remaining on the stack: ((({{[(here)

```
You entered the file:
TEST.txt

ERROR - Brace(s) still on stack: [{{(((

-- program is finished running --
```

*Note that the remaining braces on the stack are printed in reverse order (i.e. they are popped off the stack one at a time). This check should only be done if the entire input has been processed and every closing bracket found a match. **You should NOT print this error if there was an earlier brace mismatch.**

For full credit, **the output should match this format exactly**. Take note of the:

1. Exact wording of the statements.
2. Program argument printed on a new line.
3. Blank line under the program argument.
4. Error or success message printed on its own line.
5. Blank line under the final result message.

Functionality

The functionality of your program will be as follows:

1. Read a file name from the program arguments (file must be **located in the same directory** as MARS.jar file in order for the read to work. **Just drag and drop the jar file into the Lab4 folder, but do not push the jar to the remote repo!**).
    - Valid names use ASCII characters a-z, A-Z, 0-9, period and underscore (You can assume no spaces!). They must start with a letter.
    - Only accept program argument input with a max of 20 characters including the suffix.
2. Print the user’s input.
3. Open the file and read through a buffer of characters at a time.
    - The buffer should be 128 bytes of statically allocated memory (i.e. in the “data” segment).
    - Your program should iterate through the buffer and decide whether to push or pop from the stack (or do nothing) for each character.
    - Your program should counts, brackets and parentheses.
    - When the buffer is empty, it should read again from the file and repeat unless the end of the file is reached.
4. You should print an **ERROR** message if:
    - The file name was invalid.
    - An extra or mismatched brace, bracket or parenthesis is found.
    - You reached the end of the file and there are unmatched items still on the stack.
5. You should print a SUCCESS message if:

    - The end of the file is reached AND the stack is empty AND there were matching closing braces, brackets and parentheses in the correct order for every opening one.

6. Exit the program cleanly using syscall 10.

Pseudocode

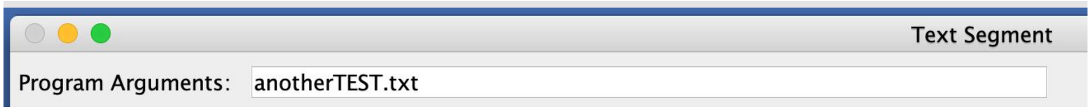
You should create pseudocode that outlines your program. Your pseudocode will appear underneath the header comment in Lab4.asm. Guidelines on developing pseudocode can be found here: https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

You may modify your pseudocode as you develop your program. **Your pseudocode must also be present in your final submission.**

Program Arguments

Your code must read and print the program argument to the screen. For example:

Input



Program Arguments: anotherTEST.txt

Output

```
You entered the file:
anotherTEST.txt

SUCCESS: There are 0 pairs of braces.

-- program is finished running --
```

Automation

Note that part of our grading script is automated, so **it is imperative that your program's output matches the specification exactly**. Output that deviates from the spec will cause point deduction.

Your code should end cleanly without error. **Make sure to use the exit syscall** (syscall 10).

Files

Lab4.asm

This file contains your pseudocode and assembly code. Follow the code documentation guidelines here.

README.txt

This file must be a plain text (.txt) file. It should contain your first and last name (as it appears on Canvas) and your CruzID. Instructions for the README can be found here.

test1.txt test2.txt test3.txt

You should create three of your own test cases named as above. These should be tests for success and errors!

Google Form

You are required to answer questions about the lab in this Google Form. Answers, excluding the ones asking about resources used and collaboration should total at the very least 150 words.

Syscalls

When printing the integer values, you may use syscall 1 (print integer). When opening and reading a file, you should use syscall 13 (open file), 14 (read file) and 16 (close file). For **syscall 13** set the mode register to a value of 0 (**$a2=0**, the mode register is ignored in MIPS). For the flag register ($a1), read the documentation provided in the help section of MARS to figure out what value to set it to (hint want to set it to read).

File that is read must be **located in the same directory** as MARS.jar file.

You should close your file when you are done using syscall 16.

Make sure to exit your program cleanly using syscall 10.

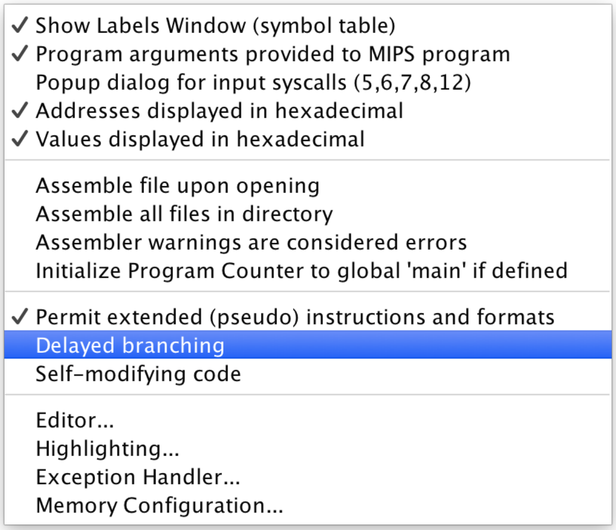
Note

It is important that you **do not hard-code** the values for any of the **addresses** in your program.

Other Requirements

Turn Off Delayed Branching

From the settings menu, **make sure Delayed branching is unchecked**



✓ Show Labels Window (symbol table)
✓ Program arguments provided to MIPS program
Popup dialog for input syscalls (5,6,7,8,12)
✓ Addresses displayed in hexadecimal
✓ Values displayed in hexadecimal

Assemble file upon opening
Assemble all files in directory
Assembler warnings are considered errors
Initialize Program Counter to global 'main' if defined

✓ Permit extended (pseudo) instructions and formats
Delayed branching
Self-modifying code

Editor...
Highlighting...
Exception Handler...
Memory Configuration...

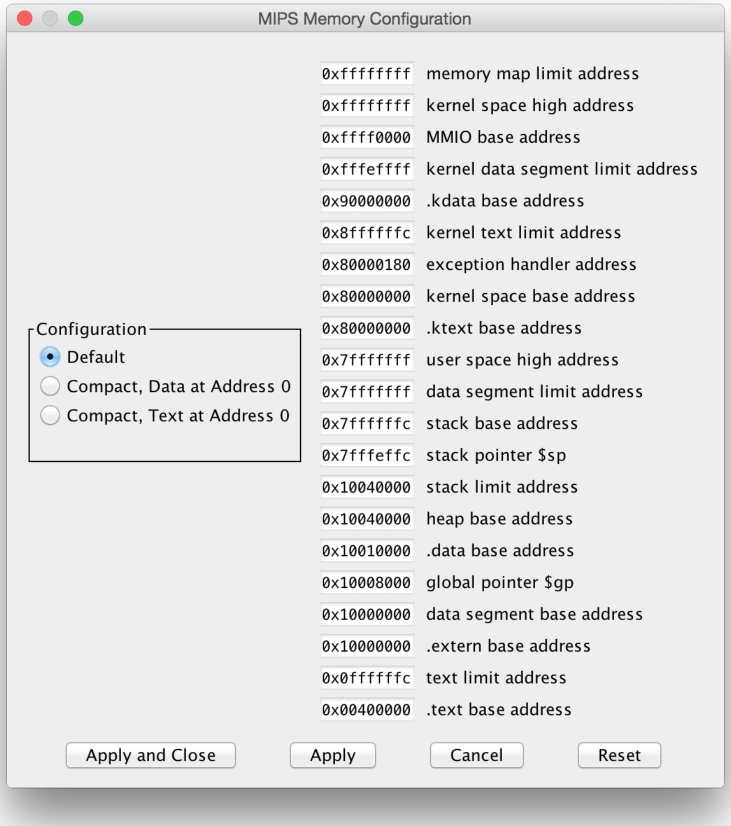
Checking this option will insert a “delay slot” which makes the next instruction after a branch execute, no matter the outcome of the branch. To avoid having your program behave in unpredictable ways, make sure Delayed branching is turned off. In addition, add a NOP instruction after each branch instruction. The NOP instruction guarantees that your program will function properly even if you forgot to turn off delayed branching. For example:

```
      LI   $t1 2

LOOP: NOP
      ADDI $t0 $t0 1
      BLT  $t0 $t1 LOOP
      NOP                     # nop added after the branch instruction
      ADD  $t3 $t5 $t6
```

MIPS Memory Configuration

To find the program arguments more easily in memory, you may choose to develop your program using a compact memory configuration (Settings -> Memory Configuration). However, your program **MUST** function properly using the **Default** memory configuration. You should not run into issues as long as you **do not hard-code any memory addresses** in your program. Make sure to test your program thoroughly using the **Default** memory configuration.



Configuration
● Default
○ Compact, Data at Address 0
○ Compact, Text at Address 0

```
0xffffffff  memory map limit address
0xffffffff  kernel space high address
0xffff0000  MMIO base address
0xfffeffff  kernel data segment limit address
0x90000000  .kdata base address
0x8ffffffc  kernel text limit address
0x80000180  exception handler address
0x80000000  kernel space base address
0x80000000  .ktext base address
0x7fffffff  user space high address
0x7fffffff  data segment limit address
0x7ffffffc  stack base address
0x7fffeffc  stack pointer $sp
0x10040000  stack limit address
0x10040000  heap base address
0x10010000  .data base address
0x10008000  global pointer $gp
0x10000000  data segment base address
0x10000000  .extern base address
0x0ffffffc  text limit address
0x00400000  .text base address
```

A Note About Academic Integrity

Please review the syllabus and look at the examples in the first lecture for acceptable and unacceptable collaboration. **You should be doing this assignment completely by yourself!**

Grading Rubric (100 points total)

```
15 pt assembles without errors
12 pt three test cases
65 pt output matches the specification
      5 pt exact wording and spacing of messages
      5 pt error check invalid file names
      10 pt reads entire file
      15 pt uses stack correctly
      5 pt detects mismatched braces
      5 pt detects mismatched brackets
      5 pt detects mismatch parentheses
      5 pt outputs correct number of matching braces, brackets, parentheses
      10 pt byte offset of mismatch is correct
```

Note: credit for this section **only** if program assembles without errors

8 pt documentation
4 pt README file complete
4 pt Google form complete with at least 150 words

-50% if program only runs in a specific memory configuration or memory addresses are hard coded

-25% incorrect naming convention

-25% for putting the MARS jar file in your repo

-100% no Google form submitted or incorrect commit ID