
Lab 3: ASCII-risks (Asterisks)



Due Tuesday, 12 May, 11:59 PM

Minimum Submission Requirements

- Ensure that your Lab3 folder contains the following files (note the capitalization convention):
 - Lab3.asm
 - README.txt
- Commit and push your repository
- Complete the [Google Form](#) with the correct commit ID of your final submission

Objective

This lab will introduce you to the MIPS ISA using [MARS](#). You will write a program with several nested loops to print variable-sized ASCII diamonds and a sequence of embedded numbers.

In addition, this lab will introduce you to several different syscalls to incorporate I/O into your program.

Resources

Read

[Documentation Standards](#)

[Introduction To MIPS Assembly Language Programming](#)

section 1.3, chapters 2, 3, and 7

Videos

[Intro to MIPS](#)

[MIPS Instructions](#)

[Registers and Memory](#)

[Intro to MARS](#)

[Registers](#)

[Hello Assembly!](#)

[Getting User Input](#)

Functionality

This program will print out a pattern with numbers and stars (asterisks).

1. It will first prompt for the height of the pattern. If the user enters an invalid input such as a negative number or zero, an error message will be printed and the user will be prompted again.
2. Each symbol(either a number or space) should be separated by a **tab (not space)**

3. Then your program should print the following pattern using numbers and stars with a tab between each of them.

Example Outputs

User inputs 0, -1, and then 5

```
Enter the height of the pattern (must be greater than 0):    0
Invalid Entry!
Enter the height of the pattern (must be greater than 0):    -1
Invalid Entry!
Enter the height of the pattern (must be greater than 0):     5
```

```
1      *      *      *      *      *      *      *      *      1
2      3      *      *      *      *      *      *      3      2
4      5      6      *      *      *      *      6      5      4
7      8      9      10     *      *      10     9      8      7
11     12     13     14     15     15     14     13     12     11
```

-- program is finished running --

For a triangle with height 3, see the figure below. Note that the '␣' character indicates a tab. Also, note that there is a newline after the triangle is finished printing before the program terminates.

```
Enter the height of the pattern (must be greater than 0):␣3

1␣*␣*␣*␣*␣1
2␣3␣*␣*␣3␣2
4␣5␣6␣6␣5␣4

-- program is finished running --
```

To receive all possible points, **the output should match the sample format exactly.**

Syscalls

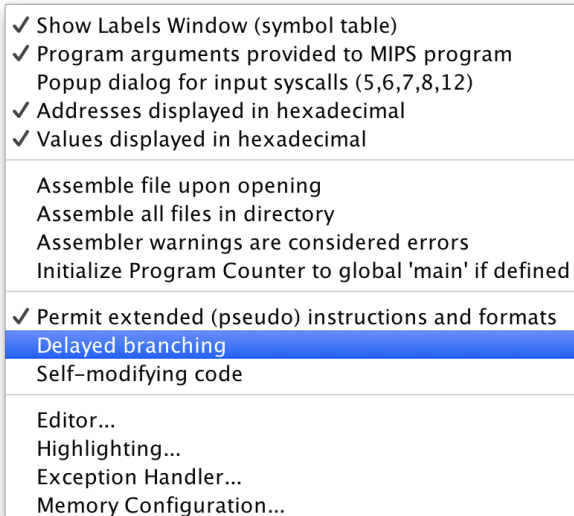
You may use **syscall 11 (print character)**, **syscall 4 (print string)** and **syscall 1 (print integer)** to print the triangle, but you must use **syscall 4 (print string)** to print the prompt message. You may collect user input using **syscall 5 (read integer)**. When finished, your program should end by calling **syscall 10 (exit)**. See below for a summary of syscalls and their uses.

SYSCALL #	FUNCTION	PURPOSE
1	Print integer	Prints an integer in ASCII
4	Print string	Print prompt
5	Read integer	Read user input
10	Exit	Successfully end the program
11	Print character	Prints an ASCII character

Table: Required Syscalls

Turn Off Delayed Branching

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



Checking this option will insert a “delay slot” which makes the next instruction after a branch executes, no matter the outcome of the branch. To avoid having your program behave in unintended 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
```

Automation

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

Files

README.txt

Instructions for the README can be found [here](#).

Lab3.asm

This file contains your pseudocode and assembly code. Include a header comment as indicated in the documentation guidelines [here](#).

Google Form

You are required to answer questions about the lab in [is Google Form](#). Question answers, excluding the ones asking about resources used and collaboration, should total at the very least 150 words.

A Note About Academic Integrity

This is the lab assignment where most students get flagged for cheating. 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 (80 points total)

20 pt assembles without errors

50 pt output matches the specification

- 5 pt exact wording and spacing of prompt (tab after colon) and error message

- 5 pt error check zero and negative heights

- 5 pt prompts user until a correct input is entered

- 5 pt correct line indentation (must use tabs)

- 5 pt number of levels match user input

- 5 pt tabs between numbers and asterisks

- 5 pt correct asterisks and numbers on each line

- 15 pt uses correct syscalls

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

10 pt documentation

- 5 pt README file complete

- 5 pt Google form complete with at least 150 words

-15 pt for incorrect naming convention

Extra credit

20 pt - Program is less than 54 instructions ([Fill Out This Form](#))