FIT1054 – Intro to Computer Science
Assessed Prac 1 – Weeks 3 and 4

Semester 2, 2018

Objectives of this practical session

To be able to write MIPS programs involving lists, local variables, and functions.
Note:

- **Local variables must be stored on the runtime stack.**
- **For all the MIPS programs you should first implement a Python version, and work out a faithful translation into MIPS. Both, the Python and MIPS code are required for your prac to be marked. However, Python solutions alone do not result in any marks, all exercises are about MIPS and MIPS solutions are required in all exercises to achieve more than 0 marks.**
- **Use only instructions on the MIPS reference sheet and use comments to document each piece of code.**
- **You may copy and paste your code between tasks where you need to reuse code.**
- **Create a new file/module for each task or subtask.**
- **Name your files task[num]_[part] to keep them organised.**

Task 1 [7 marks]

In the Gregorian calendar a *year* is a *leap year* if the *year* is divisible by 4 but not divisible by 100, or if the *year* is divisible by 400.

(a) Write a Python program `task1_a.py`, which reads in a year from the user (i.e., an integer $\geq$ 1582), and if the *year* is a leap year prints “Is a leap year”, otherwise prints “Is not a leap year”. In your module you should include a function with the signature `def is_leap_year(year)` that returns true if it is a leap year otherwise it returns false. You should avoid doing any input or output in this function, instead do the input and output as part of the **main** function.

Ex 1

```
Please enter a year: 2004
Is a leap year
```

Ex 2

```
Please enter a year: 2001
Is not a leap year
```

Run `test_task1_a.py` to ensure it works properly. Alternatively you can write your own test.

(b) Re-write your Python program (calling it `task1_b.py`) without using any user-defined functions. This will help you with a faithful MIPS translation in the following sub-question.

(c) Write a MIPS program which implements `task1_b.py`.

Task 2 [20 marks]

(a) Write a Python program `task2_a.py`, which does the following:

- Reads in the size of the `the_list` from the user.
- Reads in all the items from the user into `the_list` (you may assume they are integers).
- Prints the range of `the_list`, defined as the difference between the maximum and the minimum element in the list.

Replace any Python **for** loops for **while** loops in order to facilitate your MIPS translation.

Ex 1

```
Enter list size (positive integer): 3
Enter element 0: 9
Enter element 1: -2
Enter element 2: 3
The range of this list (maximum - minimum) is 11
```

(b) Write a Python program (`task2_b.py`) which implements `task2_a.py` without using any user-defined functions. Again, this is in preparation for the following MIPS translation.

(c) Write a MIPS program which implements `task2_b.py` faithfully including any optimisations.

Background

The bureau of climate research has a device that records the average temperature in the city for each one of the days in a month. The office is interested in understanding climate variation and has commissioned you to design a collection of programs to help analyse the data collected. The programs will run in a small portable device based on a MIPS processor. For each task, first design the algorithm to be used in Python then write the algorithms in MIPS. In each task, the input list is a list containing a number for each day in the month, one temperature record per day.
An example of one such list is:

```
26, 18, 22, 20, 13, 22, 19, 22, 20, 27, 18, 24, 15, 28, 26, 27, 20, 21, 23, 24, 27, 26, 15,
23, 22, 20, 23, 17, 18, 18
```

This list is to be read in (from the user, one element at a time) at the beginning of each task. We assume that the list will only contain natural numbers. The result of each task must be printed. The following tasks describe the functionality required.

Task 3 [20 marks]

Write some code (in *task3.py*) to find how many times a given temperature is strictly exceeded. The user should be able to enter the list (including its size), and then the target number that he/she wishes to inspect. Translate this program to MIPS faithfully. As an example, for the list given above, if the target is 25, the answer should be 7; if the target is 27, the answer should be 1, and the program would print 0 for target 28.
Ex 1

```
Enter size: 2
Enter element 0: 200
Enter element 1: 2
Comparison temperature: 15
Temperature was exceeded 1 time(s)
```

Ex 2

```
Enter size: 3
Enter element 0: 2
Enter element 1: 3
Enter element 2: 15
Comparison temperature: 15
Temperature was exceeded 0 time(s)
```

CHECKPOINT
(You should reach this point during week 3)

Task 4 [16 marks]

Write a function (in *task4.py*) that prints the frequency of each temperature in the list once. It is a good idea to decompose this problem in several functions using Python, and then do a faithful translation. Make sure you use Python structures that you know how to translate into MIPS. For example, use lists instead of dictionaries.
For example, if the input is the list given in the **Background** section, the output should be:
13 appears 1 times
15 appears 2 times
17 appears 1 times
18 appears 4 times
19 appears 1 times
20 appears 4 times
21 appears 1 times
22 appears 4 times
23 appears 3 times
24 appears 2 times
26 appears 3 times
27 appears 3 times
28 appears 1 times
You are encouraged to use re-use functions you have written for other tasks in this assignment.

*Note: you do not **HAVE** to sort the list or output however you should have **only one** instance of each count (e.g. **NOT** '24 appears 2 times', '21 appears 1 times', '24 appears 2 times') and this will be **easiest** to check if results are **sorted***

Task 5 [16 marks]

Translate `task_1a.py` into MIPS using the function calling convention studied in the class.

Task 6 [25 marks]

This section is for marks associated with optimisations you've made as part of this assignment. If you have not already included **at least two different** optimisations in your code thus far, you should do so now[1]. Prepare a report called *optimisationRationale.pdf* which responds to the following questions for **two** optimisations you have included:

[1] make sure you keep a copy of the unoptimised version in this case and submit both to moodle

1. What you were optimising for and in what way
2. Why the optimisation is appropriate for its context
3. How this changes time (or space) complexity or actual time/space taken during runtime (eg. the function was previously linear and is now logarithmic or it has reduced the memory usage by a factor of 3, etc.)

Note: some optimisations cannot be performed in python, in such cases, do the best approximation you can in python and implement the optimisation correctly in MIPS; faithfulness will be ignored for these specific cases