FIT1008: Introduction to Computer Science (FIT2085: for Engineers)

Interview Prac 1 - Weeks 3 and 4

Semester 2, 2019

Learning objectives of this practical session

To be able to write MIPS programs involving decisions, while and for loops, lists, local variables, and functions.

Important:

- **Local variables must be stored on the runtime stack.**
- **All your MIPS code needs to be properly commented.**
- **Use only MIPS instructions that appear in this semester's MIPS reference sheet.**
- **Use the function names provided, as we will use testing harnesses that rely on the MIPS code for the functions to be labeled with those names.**
- **A detailed description of what we mean by "faithfully" is given in the appendix of Tute 3 (which you should do before this prac).**

Checkpoint for the end of your lab in Week 3

To reach the check point (and thus pass the hurdle) you must complete Tasks 1 and 2, as well as the commented python code required for Task 4. Thus, the Moodle submission for the checkpoint should be a zip file containing a completed (but perhaps not yet correct) `task_1.asm`, `task_2.asm`, and a properly commented `task_4.py` files. Make sure you submit before leaving the lab. Please remember that reaching the checkpoint is a **hurdle** and you will get a 0 if you do not reach it (read the pracGuide document for more details).

While you need to submit a complete version of these tasks in week 1, you will be able to improve it during week 4 and re-submit them together with the rest of the prac in week 4, since their correctness and code quality will only be marked then. Nonetheless, complete (but possibly incorrect) code for the checkpoint tasks needs to be submitted in week 3 for you to get any marks.

After the first 30 minutes of your lab in week 4, your demonstrator will interview you to assess your level of confidence with your code, and (if there is time) will also asses its correctness and quality. If there is no time, the marking will occur outside the lab. Please make sure you zip all required files and submit them before leaving your lab. Otherwise, you will get a 0 for this prac.

Task 1 [15 marks]

Consider the following uncommented Python code:

```
a = 4
b = 5

if a > 0 and a <= b:
    result = b//a
elif a == b or b < 0:
    result = a*b
else:
    result = b//2

print(result)
```

which prints 1, since **a**>0 and **a**<=**b**, which means **b**//**a** is executed (**result** = 5//4 = 1). Faithfully translate the above code into a properly commented MIPS program assuming both **a** and **b** are global variables, and store it in file `task_1.asm`. You can easily test it by changing the values of **a** and **b**. In addition, we have provided you with a simple test harness for the if-elif-else part in file `test_task1.asm`. To add it to your code you will need to:

1. Substitute your .**text** assembly directive by the code in `test_task1.asm`.
2. Add the following two lines to your code, right after printing the result and before the syscall with code 10 that would exit your program

```
jr $ra

exit:
```

When you run your code with the above modifications it should print:

```
4
1
6
```

You are very welcome to extend the testing harness to try other combinations of **a** and **b**. We will indeed have an extended testing harness that we will use to mark you.

Task 2 [25 marks]

The following uncommented Python code reads a list of integers of the size given by the user and computes (and prints) the minimum element in the list:

```
size = int(input("Enter list size: "))

the_list = [0] * size

for i in range(size):
    the_list[i] = int(input("Enter element "+ str(i) + ": "))

if size > 0:
    min = the_list[0]
    for i in range(1,size):
        item = the_list[i]
        if min > item:
            min = item

    print( "The minimum element in this list is " + str(min) + "\n ")
```

Faithfully translate the above code into a properly commented MIPS program assuming all variables are global, and store it in file `task_2.asm`. To make your life simpler, we have provided you with the begining of the code in file `task_2.asm`. Please complete it (without changing the variable names) and ensure it runs properly.

Again, we have provided you with a test harness for the computation of the min part (you need to test the read part by hand) in file `test_task2.asm`. To add it to your code you will need to: add the code in `test_task2.asm` to your `task_2.asm` file right before the **exit**: label. When you run your code with the above modifications it should (after you manually enter a list and it prints its minimum) print:

```
The minimum element in this list is -1
The minimum element in this list is 2
The minimum element in this list is 0
```

Again, you are very welcome to extend the testing harness to try other possibilities, as we will do when marking your code.

Task 3 [15 marks]

Write in file `task_3.asm` a modification of your MIPS code in Task 2 to make it functional. That is, to faithfully translate the following Python code into a properly commented MIPS program:

```
def read_list():
    size = int(input("Enter list size: "))
    the_list = [0] * size

    for i in range(size):
        the_list[i] = int(input("Enter element "+ str(i) + ": "))
    return the_list

```

```
def get_minimum(the_list):
    size = len(the_list)
    if size > 0:
        min = the_list[0]
        for i in range(1,size):
            item = the_list[i]
            if min > item:
                min = item
        return min
    else:
        return 0 # this should return an exception (see later in the course)

def main():
    my_list = read_list()
    print(get_minimum(my_list))
```

You can copy and paste the part of the MIPS code you wrote in Task 2 that is useful for this one. However, **make sure your variables now are all local to the appropriate function (that is, no global variables are now allowed).**

Extend your `task_3.asm` file to include testing for the **get_minimum** function (ensure you have at least the same cases as we gave you in Task 2). **Important: Make sure you use as labels for the functions the names we provided** (that is, **main**, **read_list** and **get_minimum**), since those are the ones we will use with our test harness.

Task 4 [25 marks]

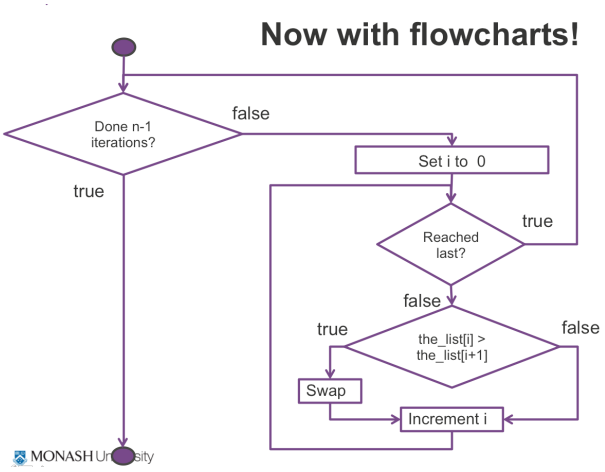
The following uncommented Python code:

```
def bubble_sort(the_list):
    n = len(the_list)
    for a in range(n-1):
        for i in range(n-1):
            item =  the_list[i]
            item_to_right = the_list[i+1]
            if item > item_to_right:
                the_list[i] = item_to_right
                the_list[i+1] = item
```

defines function `bubble_sort(the_list)`, which sorts `the_list` in increasing order, using the following algorithm:

Assume the size of `the_list` is n. We will iterate $n-1$ times. In each of these iterations, we will start traversing `the_list` from position $i = 0$ and then do the following: while $i < n-1$

- Compare each item in `the_list[i]` to the item on its right `the_list[i+1]`
- If `the_list[i]` > `the_list[i+1]`, we swap them, otherwise we do not swap
- Increment i



(a) Write a `task_4.py` that contains the above python function properly commented. Our main aim here is to ensure you understand the code. Run it with a few lists and see how it works.

(b) Write a properly commented MIPS program `task_4.asm` that faithfully implements `task_4.py` and includes testing for the **bubble_sort** function. **Important: Make sure you use as label the bubble_sort** name, since that is the one we will use with our test harness.