AUT UNIVERSITY COMPUTER + MATHEMATICAL SCIENCES

COMPUTER-BASED MID-SEMESTER TEST

2015 Semester 1

PAPER DESCRIPTION: Programming 1/Programming for Engineering Applications

PAPER CODE: 405701/735318

TIME ALLOWED: 1.5 Hours plus 5 Minutes Reading Time

TOTAL MARKS: 70

INSTRUCTIONS:

1. This test is held under exam conditions - do not talk after you have entered the examination room. Raise your hand and wait for an invigilator if you need assistance
2. Set up your Raspberry Pi at a free monitor
3. Login to your Raspberry Pi as you usually would
4. If you have not already synced and received the file **`~/p1.2015s1/test01/test1.zip`** do so now
5. Navigate to the directory **`~/p1.2015s1/test01/working_copy`**
6. Unzip the test materials using the command **`unzip -P <password> test1.zip`** - the password will be provided by the exam invigilator
7. **Do not start to read or answer questions until told to do so by an invigilator**
8. **After completing the test you must sync before leaving. Do not leave until given permission to do so by an invigilator.**

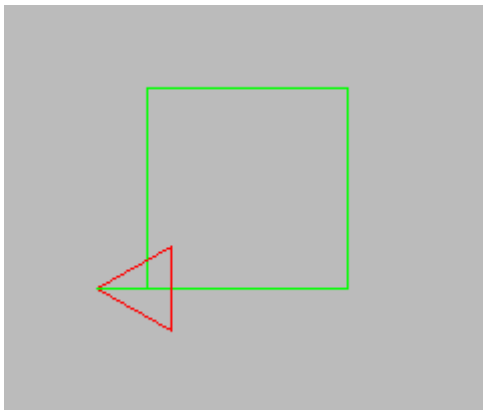
ADDITIONAL MATERIALS:

1. You may refer to your own notes and printed materials, including books
2. You may not use any electronic devices except for your Raspberry Pi. Turn off mobile phones.
3. During the test you should not attempt in any way to communicate outside the exam room, except via sync.

Q1 [10 marks]

Navigate to the directory **`~/p1.2015s1/test01/working_copy/q01`**

In file **`test01q01.c`** add turtle world function calls to the **`main()`** function to draw the following image of a green square with sides 100 pixels long.



You can test your program using the **`make`** command followed by **`./test01q01`**

If **`make`** produces errors relating to file timestamps due to the clock on your Pi not being set use the following commands to correct the problem before rebuilding. This will work for all questions.

```
touch *
make clean
```

For full marks your code must

- Create the correct output
- Be correctly indented

Q2 [10 marks]

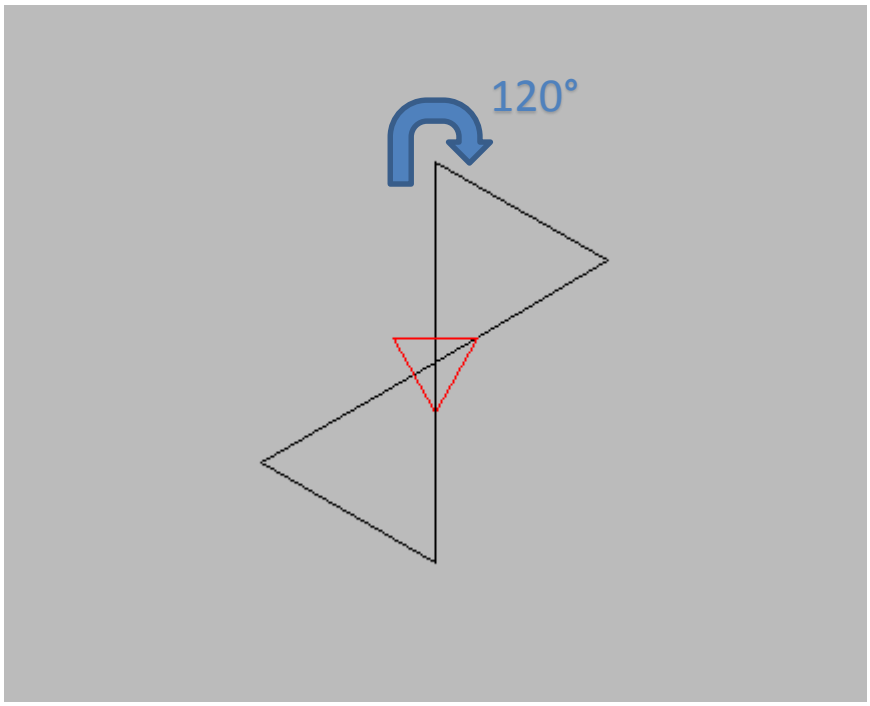
Navigate to the directory **`~/p1.2015s1/test01/working_copy/q02`**

In file **`test01q02.c`** define a function with the following signature…

```
void draw_triangle()
```

…that will draw a single equilateral triangle with sides 100 pixels long.

In the function **`main()`** add calls to **`draw_triangle()`** and any extra turtle commands necessary to draw the following image. An angle is marked on the image in blue to help you – you do not need to draw the blue markings.



For full marks your code must

- Create the correct output
- Be structured as described in the question
- Be correctly indented

Q3 [10 marks]

Navigate to the directory **`~/p1.2015s1/test01/working_copy/q03`**

In file **`test01q03.c`** write a computer program that will output exactly the following on the console.

```
1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
5 * 2 = 10
6 * 2 = 12
7 * 2 = 14
8 * 2 = 16
9 * 2 = 18
10 * 2 = 20
```

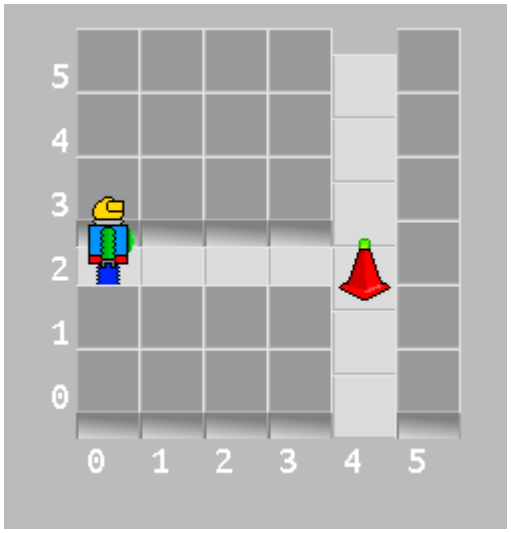
For full marks your code must

- Use a single **`for`** loop
- Define only one local variable
- Be correctly indented

Q4 [10 marks]

Navigate to the directory **`~/p1.2015s1/test01/working_copy/q04`**

In this robot world problem the robot starts at the west of the world and must walk east along a corridor until it meets a wall.



There may be a beeper in front of the wall - if there is the robot should pick it up, turn left, and walk to the end of the north corridor, then drop the beeper.

Otherwise the robot should turn right and walk to the end of the south corridor.

In the file **`test01q04.c`** write a function called **`move_robot()`** that implements this algorithm.

To run your program type

`make test`

Press **`ESC`** to quit after each test runs.

The corridors will always be the same length.

For full marks your code must

- Pass all tests
- Use at least one loop
- Be correctly indented

Q5 [10 marks]

Navigate to the directory **`~/p1.2015s1/test01/working_copy/q05`**

This is a harder version of Q4. The robot starts at the west of the world and must walk east until it finds a stack of beepers. It should count how many times it moves forwards to find the beepers. It must pick up all of beepers and count how many there are. This time the lengths of the corridors and the number of beepers in the stack will vary.

If the distance to the stack is more than 4 spaces or the number of beepers less than 4 it should turn left, move to the end of the north corridor, and drop all of the beepers.

Otherwise, if the number of beepers is 5 then the robot should turn right, move to the end of the south corridor, and drop all of the beepers.

Otherwise the robot should turn around, move back to the end of the west corridor and drop all of the beepers.

In the file **`test01q05.c`** write a function called **`move_beepers()`** that implements this algorithm. Note the name of the function is different from Q4.

Your implementation should also define and use the following functions:

`pick_up_stack()`

`move_to_wall()`

`drop_all_beepers()`

For full marks your code must

- Pass all tests
- Be structured as described in the question
- Make appropriate use of local variables and return values
- Be correctly indented

Q6 [10 marks]

Navigate to the directory **`~/p1.2015s1/test01/working_copy/q06`**

In file **`test01q06.c`** write a function called **`draw_text_box()`** that will produce output such as the following.

```
+---+
|   |
+---+

OooooO
x     x
x     x
OooooO

#============#
H            H
H            H
H            H
#============#
```

The function should take 5 parameters: the character to use for the horizontal sides; the character to use for the vertical sides; the character to use for the corners; the width of the box in characters; and the height.

Define at least two helper functions to structure your algorithm.

The starting file contains a **`main()`** function with a number of calls to the function to help you test it. You can also add your own calls.

For full marks your code must:

- Reliably generate the correct output
- Be structured as described in the question
- Use meaningful names for functions and variables
- Be correctly indented

Q7 [10 marks]

Navigate to the directory **~/p1.2015s1/test01/working_copy/q07**

In file **test01q06.c** write a program to convert lengths between different units. A sample dialog should look something like this:

```
Hello!
Please enter the number of units that you want to convert.
> 17.5
What is the input unit?
> x
I don't recognise that unit.  I can handle:
" - inches
' - feet
c - centimeters
m - meters
What is the input unit?
> '
What is the output unit
> m
17.5' is 5.34m
```

The following table of conversion factors will be needed:

		to			
		inches (")	feet(')	centimeters (c)	meters(m)
from	inches (")	1.00	0.0833	2.54	0.0254
	feet (')	12.0	1.00	30.5	0.305
	centimeters (c)	0.394	0.0328	1.00	0.00100
	meters (m)	39.4	3.28	100	1.00

Make sure that your program can reliably the user entering an incorrect unit, even multiple times (you do not have to handle the user entering an incorrect length). Structure your program appropriately using multiple functions, and comments where necessary to make your algorithm clear.

This question is potentially both moderately difficult and time consuming so marks will be awarded for partial progress.

You may find the man pages for **printf()** and **scanf()** useful. At the command line use

```
man printf
```

…or…

```
man scanf
```

…to read these.

For full marks your code

- Reliably generate the correct output
- Be structured as described in the question
- Use meaningful names for functions and variables
- Include helpful comments
- Be correctly indented

[END OF TEST]