Week 03, Homework 03	Weight: 2%	Due: Monday week 5, 09:00am (via `sync`)

Pre-homework Preparation:

- **Lab: Week 03, Lab 03**
- **Lectures: Weeks 01, 02, 03**

Homework Activities:

Obtain the homework files using **`sync`**.

For each question's source file, be sure to fill in the file comment header accurately!

Question 1: Robot basics

When writing robot programs it usually helps to write some simple functions to enhance the capabilities of the robot.

Navigate to the directory: **`~/p1.2015s1/homework03/working_copy/q01/`**

Open **`hw03q01.c`** and define dummy functions with the following signatures:

```
void turn_robot_around();

void turn_robot_right();

void pick_up_stack();

void drop_all_items();

void move_robot_forwards_n(int n);
```

Note that dummy functions that take arguments (like the last one above) need to define their parameters. So the dummy function for move_robot_forwards_n() will look like this:

```
void move_robot_forwards_n(int n)
{
}
```

Now do **`make test`** and read the output from the tests. You will also find it helpful to read the tests in **`hw03q01_test.c`**.

Fairly obviously the first function should turn the robot around 180°, the second should turn the robot right, the third should pick up however many beepers are present on the ground at the robot (or none if there are none) the fourth should drop all of the items that the robot is holding (or none if holding none) and the last should move the robot forwards the specified number of times. Complete the implementation of these functions and ensure that all tests pass.

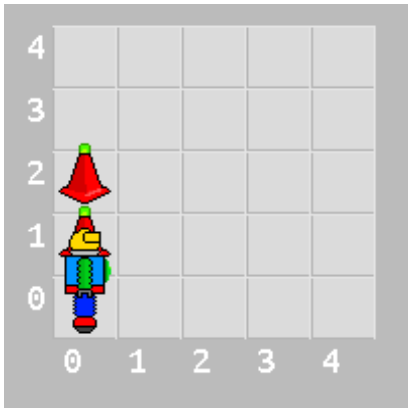
Question 2: Beeper transporter

Navigate to the directory: **`~/p1.2015s1/homework03/working_copy/q02/`**

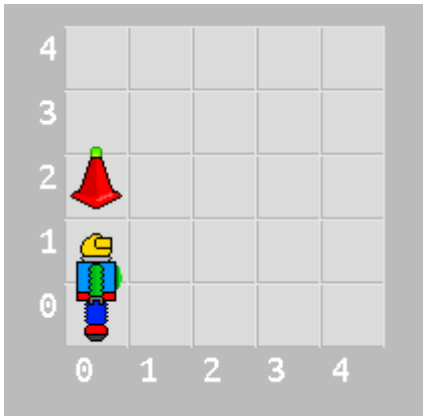
Edit the file hw03q02.c and create a dummy function with the following signature:

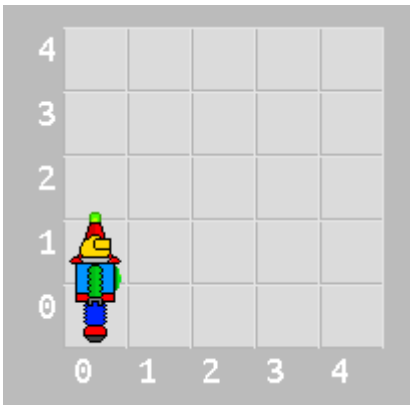
```
void move_beepers();
```

In this simple exercise the robot must transport two beepers from the east to the west of the world. The beepers start at coordinates (0, 1) and (0, 2) and must be placed in a stack at (4, 2).



However, one or more of the beepers may be missing at the start of the scenario. If either beeper is missing then the robot should leave them in their original locations and return to its starting position.





First plan your algorithm. You may find it helpful to draw a flowchart.

It is permissible to pick up a beeper then return it to its original position if necessary.

Try to avoid making the robot perform more actions than necessary.

Do not use any variables. They are not necessary to accomplish this task.

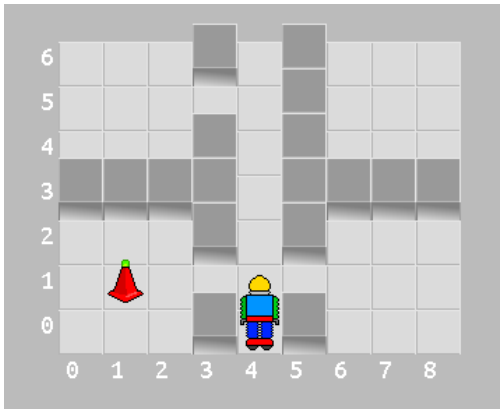
You do not need to define additional functions to help in this question, but may do if you wish.

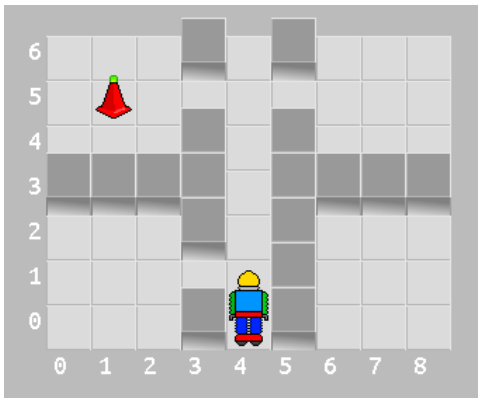
Question 3: Four rooms

Navigate to the directory: **`~/p1.2015s1/homework03/working_copy/q03/`**

Using **`less`** or **`editor`**, view the file **`hw03q03.h`** to see what function you need to define (a prototype for it is given in the header file).

In **`hw03q03.c`** define an appropriate dummy function. Run **`make test`** and view the robot world.





In the world for this question there are 4 rooms, two to the east, and two to the west, with a corridor between them. At random a beeper will appear in one of the two rooms to the west. The robot must move it to the centre of one of the rooms to the east. But only one of the doors will be open for the rooms to the east. Two of the four possible configurations are shown above

First plan your algorithm. There are a number of steps in this task, such as *move to first room*, *move to second room*, and a some decisions to be made, such as *if door is closed*…

Once you have planned your algorithm write the steps in hw03q03.c using comments.

Now complete the algorithm so that all tests pass, retaining the comments. You may need to rewrite your comments if your algorithm changes as you develop your code.

You should define additional functions for the larger steps in your algorithm, such as **`move_to_first_room()`**, and also for useful utilities such as **`turn_robot_around()`** and **`move_robot_forwards_n()`**.

Question 4: Robot sandbox

Your job for this question is to invent your own robot challenge and write the code to beat it.

In the file **hw03q04.c** you must create two functions with the following signatures

```
void setup_challenge();

void complete_challenge();
```

setup_challenge() should create a robot world for your challenge, including any walls and beepers that might be needed, and also create a robot at a starting position of your choosing. For this question the functions in the header file **/usr/local/include/p1sandbox.h** are available for you to use to do this.

complete_challenge() should control the robot to complete the challenge successfully. The test assumes only that completing the challenge leaves the robot alive (you don't want to kill the poor robot do you?).

Your challenge should require a moderate amount of programming ability to complete. Try it out on your colleagues!

Most interesting challenges require a random element – otherwise they can always be completed by the same simple sequence of robot commands. A function for generating a random integer within a range has been included in your starting code.

Document your challenge in the comment at the top of **hw03q04.c**. Clearly describe the scenario that is presented, and any success or failure criteria.

You might find this question easier to complete once you have tried Question 5, which also uses functions from **p1sandbox.h**.

Question 5: Build a maze, then escape it!

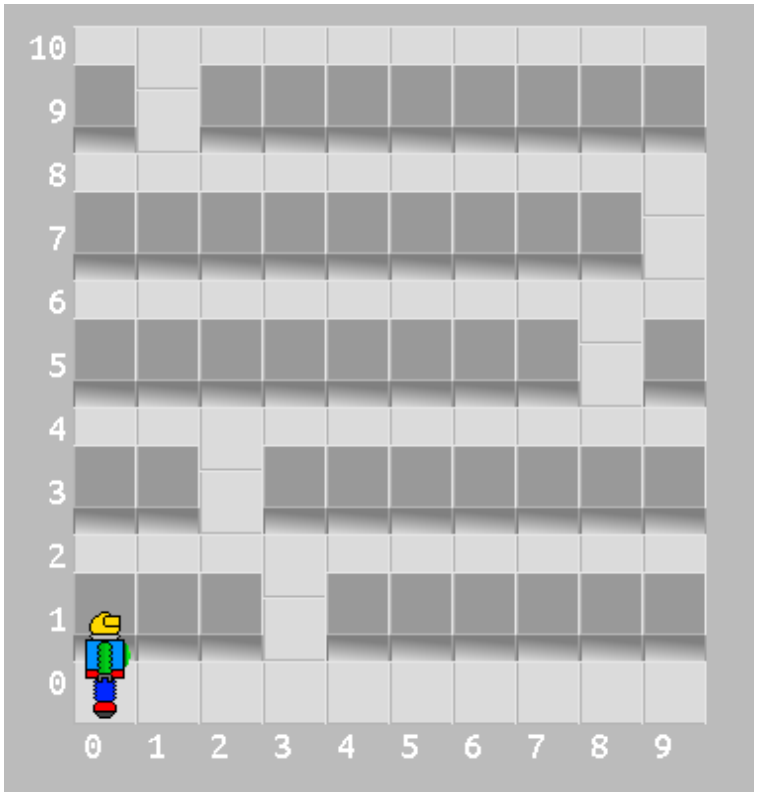
This question involves again involves the use of the robot sandbox functions to set up a challenge, and then the robot functions to solve the challenge.

In the file **`hw03q05.c`** you must create two functions with the following signatures

```
void setup_maze();

void complete_maze();
```

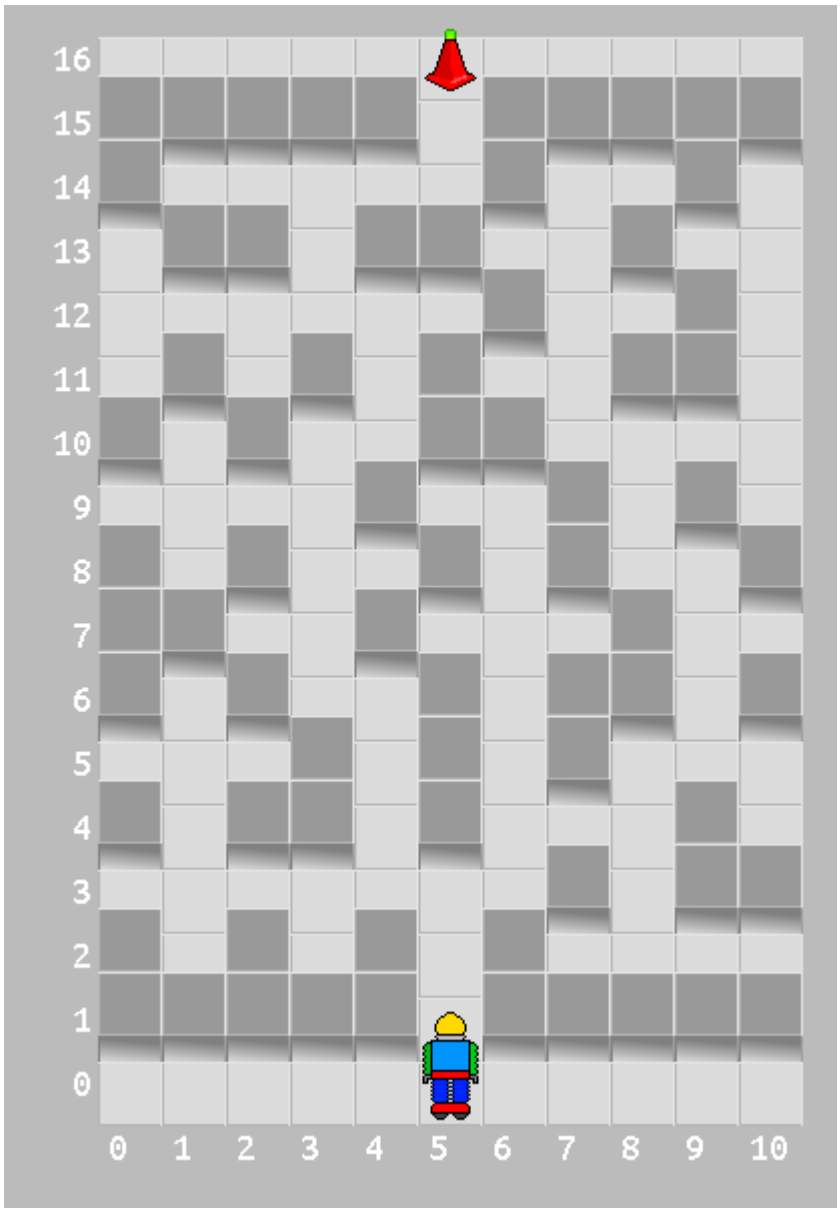
This time the robot world is created in the test. The function **`setup_maze()`** which you define should create a sequence of west-east walls as shown below. Each wall has a single gap in a random location along its length.



The test will then create a robot at the south of the world. In **`complete_maze()`** write code to navigate the robot to the top of the maze.

Question 6: Escape the maze 2

This question is easier than it seems. In hw03q06 write code to move the robot from the south to the north of the robot world, passing through a maze such as that shown below. The maze is generated randomly each time, but is guaranteed to contain *no internal circuits*. You may wish to google *maze solving algorithms*. In fact this challenge can be solved without the use of variables. Note that finding the beeper can be used to check whether the robot has escaped the maze.



Homework Submission:

Run the **`sync`** command to submit your completed homework.

Marking Criteria:

Have you completed each of the following?

Marking Criteria:	Week 03, Homework 03 Weight 2%	Maximum Possible Mark:	Mark achieved?
Q1:	Code correct an all tests pass?	8	
	Correct code layout?	2	
Q2:	At least one test passes?	3	
	All tests pass?	5	
	No variables used?	1	
	No unnecessary steps taken by robot?	1	
Q3:	At least one test passes?	3	
	All tests pass?	3	
	Algorithm outlined using comments?	2	
	Appropriate use of functions to structure algorithm?	2	
Q4:	Setup and complete functions work and pass challenge?	4	
	Interesting challenge requiring moderate programming ability (loops, conditional statements, functions) to solve?	4	
	Challenge clearly documented in comment?	2	
Q5:	setup_maze() sets up maze correctly?	4	
	complete_maze() reliably solves maze?	4	
	Evidence of algorithm planning and structuring into functions?	2	
Q6:	Algorithm reliably solves maze.	8	
	Good commenting practices followed?	2	
Total:		**60**	