Zeid Kootbally
Fall2022
UMD
College Park, MD

RWA1 (v0.0.0)

ENPM809Y: Introductory Robot Programming

Due date: **Friday, September 30, 2022, 5 pm**

Contents

1 Updates

This section describes updates added to this document since its first released. Updates include addition to the document and fixed typos. The version number seen on the cover page will be updated accordingly. NOTE: There may be some mistakes even after proofreading the document. If this is the case, please let me know.

- **v0.0.0**: Original release of the document.

2 Conventions

In this document, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

- This is a [file.txt]
- This is a `folder`
- Terminal outputs will be displayed as follows:

Listing 2.1: Output example.

```
hello, world
hello, world
hello, world
```

3 Introduction

This assignment must be performed individually. Exercises for this assignment revolve around a user providing instructions to a robot through the terminal.

- This assignment consists of 2 exercises.
- Each exercise must be implemented in a separate [*.cpp] file.
  - Exercise#1 must be implemented in [exercise1.cpp] and contains of a `main()` function.
  - Exercise#2 must be implemented in [exercise2.cpp] and contains of a `main()` function.
- Materials needed to complete this assignment can be found in the lecture slides, including RM1 and RM2 slides.

TODO: Create a folder `rwa1_firstname_lastname` and create two files in this folder (see Figure 1 as an example). When you are done with this assignment, compress the whole folder (.zip, .tar, .rar, etc) and upload it on Canvas.

NOTE: Some of you may be tempted to perform this assignment with functions and maybe with object-oriented programming (OOP). However, we need to be fair and

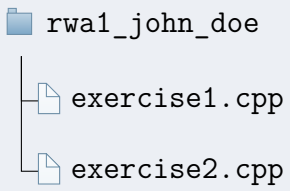

Figure 1: Folder structure example.

grade everyone the same way. Therefore, the use of functions (except `main()`) and OOP are prohibited. Moreover, function and OOP implementations have strict best practices which we will see in future lectures. If you still want to use functions and/or OOP and you do not implement these best practices then you will be graded accordingly and you will be penalized. You will be given more freedom starting from the next assignment.

NOTE: It is highly recommended you read this document 2 or 3 times before you start working on the assignment. There is a lot of information and you may miss some of it on your first read.

4 Exercise #1

This exercise involves a robotic arm that can perform pick-and-place. The arm can pick up parts and place them on tables. The components involved in this environment are described below.

- 1 robotic arm.
- 3 parts: 1 cube (c), 1 ball (b), and 1 peg (p).
- 2 grippers: 1 suction gripper (s) and 1 finger gripper (f).
- 3 tables: 1 red table (r), 1 green table (g), and 1 blue table (b).

The rules of pick-and-place are as follows:

- The arm can only pick up the ball with the suction gripper.
- The arm can only pick up the cube and the peg with the finger gripper.
- The cube can be placed only the red table, the ball only on the green table, and the peg only on the blue table.

TODO: Your task for this exercise is to ask the user to enter the following information in the specified order:

1. The object to pick up.
2. The gripper to use.
3. The table to place the object.

Based on the user inputs, your program will output a success or a failure message. An example of outputs from the program is shown in Listing 4.1.

Listing 4.1: Exercise#1 – Example of outputs.

```
---------------------------------
EXERCISE 1
---------------------------------
Which part to pick up (c, b, p)? p
---------------------------------
Which gripper to use (s, f)? s
---------------------------------
Failure: Cannot pick up the peg with the suction gripper.
---------------------------------
Try again (y, n)? y

---------------------------------
EXERCISE 1
---------------------------------
Which part to pick up (c, b, p)? p
---------------------------------
Which gripper to use (s, f)? f
---------------------------------
Which table to use (r, g, b)? g
---------------------------------
Failure: Cannot place the peg on the green table.
---------------------------------
Try again (y, n)? y

---------------------------------
```

```
EXERCISE 1
-----------------------------------
Which part to pick up (c, b, p)? p
-----------------------------------
Which gripper to use (s, f)? f
-----------------------------------
Which table to use (r, g, b)? b
-----------------------------------
Success: The peg was placed on the blue table.
-----------------------------------
Try again (y, n)? n
```

Important points to implement in your program:

- This example shows 3 attempts with the first 2 attempts resulting in a failure and the third attempt being a success. Successes or failures are based on the pick-and-place rules provided above.
  - On the first attempt (starts at line 4), the user selected the peg and the suction gripper. Based on the rules, the peg cannot be picked up by the suction gripper and a failure message was displayed.
  - On the second attempt (starts at line 15), the user selected the correct gripper for the peg but the table was not the correct one. Another failure message was displayed.
  - On the third attempt (starts at line 28), the user selected the correct gripper and the correct table for the peg. In this case a success message was displayed.
- Failure messages are displayed as soon as the rules are broken. In the first attempt, the failure message was displayed as soon as the wrong gripper was selected for the peg. The program did not even ask about the table.
- Failure and success messages use full object names, gripper names, and table names. For instance, we displayed “Failure: Cannot place the peg on the green table" and not “Failure: Cannot place p on g".
- Finally, after a failure or a success, the program asks the user if they want to try again.

4.1 Challenges

This exercise does not contain big challenges but requires heavy uses of selections and iterations.

- One of the challenges consists of restarting the program when the user wants to try again.
- Another challenge is to ensure the inputs the user provides are correct. If the user does not enter correct inputs, the program should re-prompt the same question again until correct inputs are provided. An example of program outputs with incorrect inputs is provided in Listing 4.2. NOTE: An example was provided in

RM2 on how to address incorrect inputs with a *do-while statement*.

Listing 4.2: Exercise#1 – Example of incorrect inputs.

```
-----------------------------------
EXERCISE 1
-----------------------------------
Which part to pick up (c, b, p)? a
Which part to pick up (c, b, p)? 1
Which part to pick up (c, b, p)? c
-----------------------------------
Which gripper to use (s, f)? y
Which gripper to use (s, f)? p
Which gripper to use (s, f)? f
-----------------------------------
Which table to use (r, g, b)? c
Which table to use (r, g, b)? r
-----------------------------------
Success: The cube was placed on the red table.
-----------------------------------
Try again (y, n)? a
Try again (y, n)? n
```

4.2 Grading Rubric

- This exercise is worth 10 pts.
- Your program will be tested with different input combinations.
- Incorrect inputs will be used to test the robustness of your program.
- We will look at the use of best practices seen in class, e.g., code commented, variables initialized using uniform initialization, good identifiers, etc.

5 Exercise #2

In this exercise a robot places pegs in slots based on user inputs. The components involved in the environment are described below and can be visualized in Figure 2.

- 1 robotic arm.
- 1 box with 9 slots.
- 9 red pegs (r).
- 9 green pegs (g).
- 9 blue pegs (b).

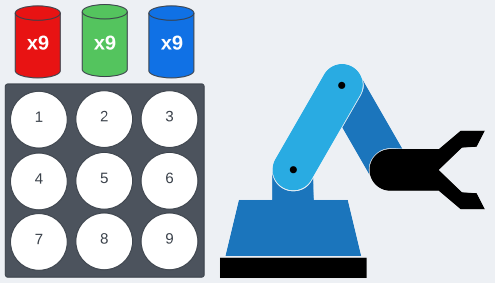

Figure 2: Components in Exercise#2.

TODO: Write a program which prompts the user for the pegs to use and their slots in the box (slots are represented with numbers as seen in Figure 2). An example of outputs for this exercise is presented in Listing 5.1.

- Use a 2D array to keep track of available and occupied slots. The 2D array must be implemented with `std::array` and it must be a variable local to your function `main()` (not a global variable). Each time the user makes a selection, this array is updated. Each time the slots are displayed in the terminal (e.g., lines 11-13 in Listing 5.1), it is from a *for statement*.
- Information on pegs is asked in this order: 1) red pegs, 2) green pegs, and 3) blue pegs.

Important points to implement your program:

- The program starts by displaying available slots (e.g., lines 4-6). The numbers on lines 4-6 should match the slots configuration seen in Figure 2.
- The user is then asked to enter whether or not they want the robot to place pegs of a given color. If the user enters **y** (e.g., line 8), then the program prompts about the slots for placing the pegs (e.g., line 9). If the user enters **n** (e.g., line 15) then the program skips the question on the slots.
- The question regarding slot selections is "dynamic". For instance, on line 9, all the slots are available and thus the program proposes the range **1-9** as a selection. On line 22, the available slots are **1, 2, 4, 5, 7, 8** since slots **3, 6, 9** are not available anymore (they were selected on line 9).
- Each time information on pegs is entered, the program displays the status of the slots. Unoccupied slots are numbered and occupied slots are represented with letters (**r** for red pegs, **g** for green pegs, and **b** for blue pegs).
- After all the selections are made, the program prompts the user if they want to try again (e.g., lines 28 and 58). **y** restarts the program and **n** exits the program.

Listing 5.1: Exercise#2 outputs example.

```
-----------------------------------
EXERCISE 2
-----------------------------------
1 2 3
4 5 6
7 8 9
-----------------------------------
Do you want to place red pegs (y, n)? y
Where to place red pegs (1-9)? 3 6 9
-----------------------------------
1 2 r
4 5 r
7 8 r

Do you want to place green pegs (y, n)? n
-----------------------------------
1 2 r
4 5 r
7 8 r

Do you want to place blue pegs (y, n)? y
Where to place blue pegs (1, 2, 4, 5, 7, 8)? 1 2 8
-----------------------------------
b b r
4 5 r
7 b r
-----------------------------------
Try again (y, n)? y

-----------------------------------
EXERCISE 2
-----------------------------------
1 2 3
4 5 6
7 8 9
-----------------------------------
Do you want to place red pegs (y, n)? y
Where to place red pegs (1-9)? 3 5 7
-----------------------------------
1 2 r
4 r 6
r 8 9

Do you want to place green pegs (y, n)? y
Where to place green pegs (1, 2, 4, 6, 8, 9)? 1 2 8 9
-----------------------------------
g g r
4 r 6
r g g

Do you want to place blue pegs (y, n)? y
Where to place blue pegs (4, 6)? 4
-----------------------------------
g g r
b r 6
r g g
-----------------------------------
Try again (y, n)? n
```

5.1 Full Slots

There are 9 slots in total. If all slots are filled out at some point during the program execution then no further questions should be asked and the program should terminate. In Listing 5.2, the user filled out all the slots with red pegs on line 9. Since there are no more slots left, questions regarding green and blue pegs are skipped. The inputs on line 26 filled out 3 slots and the inputs on line 33 filled out 6 slots. Since all the slots are full, questions regarding blue pegs are skipped.

Listing 5.2: Exercise#2 – Examples of full slots.

```
1  ----------------------------------
2  EXERCISE 2
3  ----------------------------------
4  1 2 3
5  4 5 6
6  7 8 9
7  ----------------------------------
8  Do you want to place red pegs (y, n)? y
9  Where to place red pegs (1-9)? 1 2 3 4 5 6 7 8 9
10 ----------------------------------
11 r r r
12 r r r
13 r r r
14
15 ----------------------------------
16 Try again (y, n)? y
17
18 ----------------------------------
19 EXERCISE 2
20 ----------------------------------
21 1 2 3
22 4 5 6
23 7 8 9
24 ----------------------------------
25 Do you want to place red pegs (y, n)? y
26 Where to place red pegs (1-9)? 1 2 3
27 ----------------------------------
28 r r r
29 4 5 6
30 7 8 9
31
32 Do you want to place green pegs (y, n)? y
33 Where to place green pegs (4, 5, 6, 7, 8, 9)? 4 5 6 7 8 9
34 ----------------------------------
35 r r r
36 g g g
37 g g g
38
39 ----------------------------------
40 Try again (y, n)? n
```

5.2 Challenges

- One of the challenges of this exercise is that you do not know beforehand how many inputs the user will provide. Although these inputs are integers, one way

to handle the inputs is to store them in one `std::string` variable, grab each substring from this variable using the white space as a delimiter, and convert each substring to an integer.

- Another challenge is to make sure user inputs are correct. For instance, if the user enters something else than what is proposed then your program should re-prompt the question until the user enters a valid answer. An example of such behavior is presented in Listing 5.3.
- Finally, your program should be able to handle inputs for questions regarding slots in any order. For instance, if the user enters **2 1 3** or **3 1 2** on line 11 in Listing 5.3, then your program should accept these answers as well.

Listing 5.3: Exercise#2 – Examples of incorrect inputs.

```
1  ---------------------------------
2  EXERCISE 2
3  ---------------------------------
4  1 2 3
5  4 5 6
6  7 8 9
7  ---------------------------------
8  Do you want to place red pegs (y, n)? a
9  Do you want to place red pegs (y, n)? y
10 Where to place red pegs (1-9)? 10
11 Where to place red pegs (1-9)? 1 2 3
12 ---------------------------------
13 r r r
14 4 5 6
15 7 8 9
16
17 ---------------------------------
18 Do you want to place green pegs (y, n)? y
19 Where to place green pegs (4, 5, 6, 7, 8, 9)? 3
20 Where to place green pegs (4, 5, 6, 7, 8, 9)? 1
21 Where to place green pegs (4, 5, 6, 7, 8, 9)? 4
22 ---------------------------------
23 r r r
24 g 5 6
25 7 8 9
26 ---------------------------------
27 Do you want to place red pegs (y, n)? n
28 ---------------------------------
29 Try again (y, n)? n
```

5.3 Grading Rubric

- This exercise is worth 20 pts.
- We will try different inputs to make sure your program is robust. We will try wrong inputs, we will enter slot numbers in any order, we will fill out the slots with red pegs only, with red and green pegs, and with red, green, and blue pegs.
- We will look at best practices, e.g., code commented, use of `at()` instead of `[]`, variables initialized using uniform initialization, good identifiers, etc.