Zeid Kootbally
University of Maryland
College Park, MD

zeidk@umd.edu

RWA-1: Program Flow Control and Functions

ENPM809Y : Fall 2021

Due **Tuesday, September 28, 2021, 7 pm** **(Section I)**

Due **Wednesday, September 29, 2021, 4 pm** **(Section II)**

Contents

1 Introduction

This is an individual assignment.
This assignment consists of taking user inputs and passing these inputs to different functions.

- **Scenario**: An industrial robot is tasked to fill up boxes with parts.
  - All parts are of the same type.
  - Boxes come in four different sizes: Small (S), medium (M), large (L), and extra large (XL).
  - The maximum number of parts for each box type is different. The maximum number of parts in smaller boxes should always be lower than the maximum number of parts in bigger boxes.
  - Here are some examples of valid and invalid number of parts for each box type.
    - Max number of parts: 5 (S), 16 (M), 20 (L), 25 (XL): Valid.
    - Max number of parts: 15 (S), 10 (M), 20 (L), 25 (XL): Invalid: S>M.
  - The boxes can not be partially filled. They are either full or empty.
- **Rules**:
  1. The robot must fill up bigger boxes before filling up smaller ones. In this case, the robot has to fill up XL boxes first, then L boxes, then M boxes, and finally S boxes.
  2. Boxes can not be partially filled. They either have to be full or empty. If a box can contain 10 parts then you can only have 10 parts or 0 part in this box.
- **Data Structure**: `struct` data structures can be very helpful to complete this assignment. Using `struct` is recommended but not enforced. Below is a more complex use of `struct`s. This example should give you a hint on how to approach this assignment.

```
struct s1 {
    unsigned int a{};
    unsigned int b{};
};

struct s2 {
    unsigned int a{};
    unsigned int b{};
};

struct rwa1 {
    s1 member1{};
    s2 member2{};
};

int main() {
    s1 var1{ 1, 2 };//initialize a and b for s1
    s2 var2{ 3, 4 };//initialize a and b for s2
    rwa1 application{var1, var2};//initialize member1 and member2 for rwa1
    std::cout << application.member1.a << '\n';
    std::cout << application.member1.b << '\n';
    std::cout << application.member2.a << '\n';
    std::cout << application.member2.b << '\n';
}
```

2 Instructions

Write the function definition for each function prototype below.

2.1 Total Number of Parts

```
/**
 * @brief Get the total number of parts to be placed in boxes
 * @return unsigned Number of parts entered by the user
 */
unsigned int get_total_parts();
```

In this function you will ask the user to enter the number of parts your robot will need to place in boxes.

```
How many parts in total?: 300
```

2.1.1 Verification

- You need to check the user entered only a positive integer.
- Re-prompt the user to enter a new number until a positive integer is entered.

2.2 Number of Boxes

```
/**
 * @brief Get the total number of boxes for each type
 * @param <data_structure> Data structure to hold box/part  information
 */
void get_total_boxes(<data_structure>& boxes);
```

In this function you need to prompt the user for the number of boxes available in the environment for each type. In the prototype you need to replace `<data_structure>` with the correct data structure. For instance, if you use the `rwa1` `struct` described previously, then replace `<daa_structure>` with `rwa1`. Once the user enters the number of boxes of each type, you need to update the correct items in this data structure.
An example of output for this function along with user inputs is provided below.

```
How many boxes for S/M/L/XL? 5 10 10 5
```

In this example, the user entered 5 small boxes, 10 medium boxes, 10 large boxes, and 5 extra large boxes.

2.2.1 Verification

- You need to check the user entered only positive integers.
- If the step above is not satisfied, re-prompt the user to re-enter these numbers.

2.3 Number of Parts per Box

```
/**
 * @brief Get the number of parts for each box type
 * @param <data_structure> Data structure to hold box/part information
 */
void get_part_per_box(<data_structure>&);
```

In this function you will prompt the user for the maximum number of parts each box can contain. Remember that the number of parts for each box type has to conform to S<M<L<XL (smaller boxes can not cotain more parts than bigger boxes). In the following prototype you need to replace `<data_structure>` with your data structure. Passing the data structure by reference will allow you to modify the content of the data structure directly inside the body of the function.
An example of output from this function along with user inputs is provided below.

```
How many parts per box for S/M/L/XL? 5 10 15 20
```

In this example: S can contain a maximum of 5 parts, M a maximum of 10 parts, L a maximum of 15 parts, and XL a maximum of 20 parts.

2.3.1 Verification

- You need to check the user entered only positive integers.
- You need to check the numbers entered are valid. That is, the number entered for S is not greater or equal to the numbers entered for M, L, and XL. The number entered for M is not greater or equal to the numbers entered for L and XL boxes. And so on.
- If the two steps above are not satisfied, re-prompt the user to re-enter these numbers.

2.4 Fill Up Boxes

```
/**
 * @brief Fill up boxes from user inputs
 * @param <data_structure> Data structure to hold box/part information
 */
void fill_up_boxes(const <data_structure>&);
```

This function takes a data structure as a `const` reference. We are passing this data structure as a reference and we only need to read it (not modify it). However, there is still a chance we may modify the content of this data structure by mistake. Therefore, we must pass it as a `const` reference to avoid any content overwrite.
An example of outputs for this function is given below. Each values seen in the outputs come from user inputs. In the line `XL box (5 x 20 max): 5 -- remaining parts: 206`:

- `XL box (5 x 5 max)` describes that we have 5 XL boxes and each box can contain a maximum of 20 parts.
- `: 5` describes the number of XL boxes used.
- `remaining parts: 206` describes the remaining parts (out of 306) after using 5 XL boxes. 206 = 306 - (5*20).

```
Boxes that can be built with 306 pegs:
-----------------------------------------------
XL box (5 x 20 max): 5 -- remaining parts: 206
L box (10 x 15 max): 10 -- remaining parts: 56
M box (10 x 10 max): 5 -- remaining parts: 6
S box (5 x 5 max): 1 -- remaining parts: 1
parts not in boxes: 1
```

Here is another example:

```
Boxes that can be built with 18 pegs:
-------------------------------------------------
XL box (5 x 20 max): 0 -- remaining parts: 18
L box (10 x 15 max): 1 -- remaining parts: 3
M box (10 x 10 max): 0 -- remaining parts: 3
S box (5 x 5 max): 0 -- remaining parts: 3
parts not in boxes: 3
```

In this example note that XL boxes were not used because the boxes have to either be completely filled or empty. They can not be partially filled. The same reasoning applies to M and S boxes.

2.5 Full Application

Running all 4 functions should give you the following outputs. See illustration in Figure 1.

```
How many parts in total? 18
How many boxes for S/M/L/XL? 5 10 10 5
How many parts per box for S/M/L/XL? 5 10 15 20

Boxes that can be built with 18 pegs:
-------------------------------------------------
XL box (5 x 20 max): 0 -- remaining parts: 18
L box (10 x 15 max): 1 -- remaining parts: 3
M box (10 x 10 max): 0 -- remaining parts: 3
S box (5 x 5 max): 0 -- remaining parts: 3
parts not in boxes: 3
```

Here is another example which is depicted in in Figure 2.

```
How many parts in total? 192
How many boxes for S/M/L/XL? 2 1 4 5
How many parts per box for S/M/L/XL? 5 10 20 23

Boxes that can be built with 18 pegs:
-------------------------------------------------
XL box (5 x 23 max): 5 -- remaining parts: 77
L box (4 x 20 max): 3 -- remaining parts: 17
M box (1 x 10 max): 1 -- remaining parts: 7
S box (2 x 5 max): 1 -- remaining parts: 2
parts not in boxes: 2
```

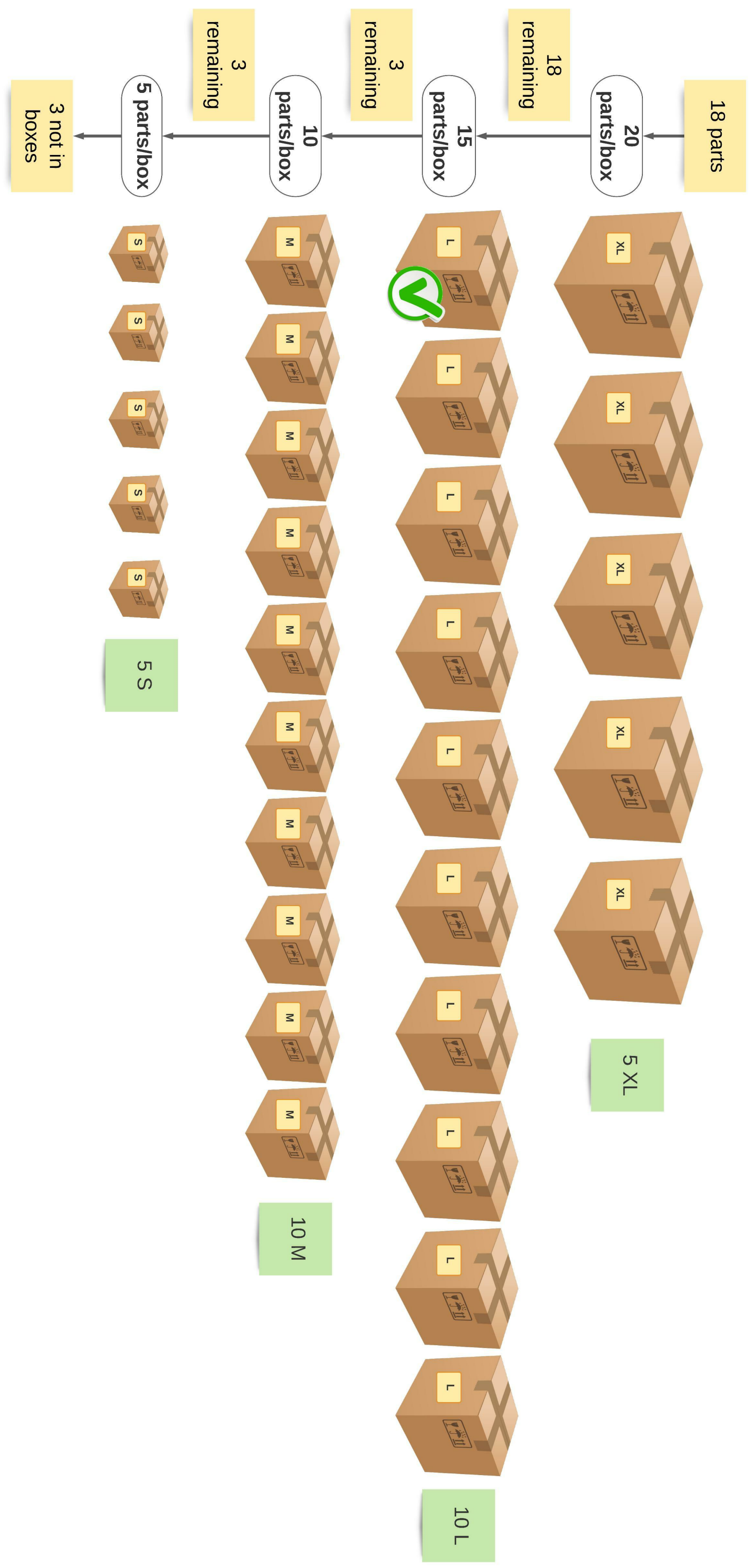

Figure 1: Filling up boxes for 18 parts.

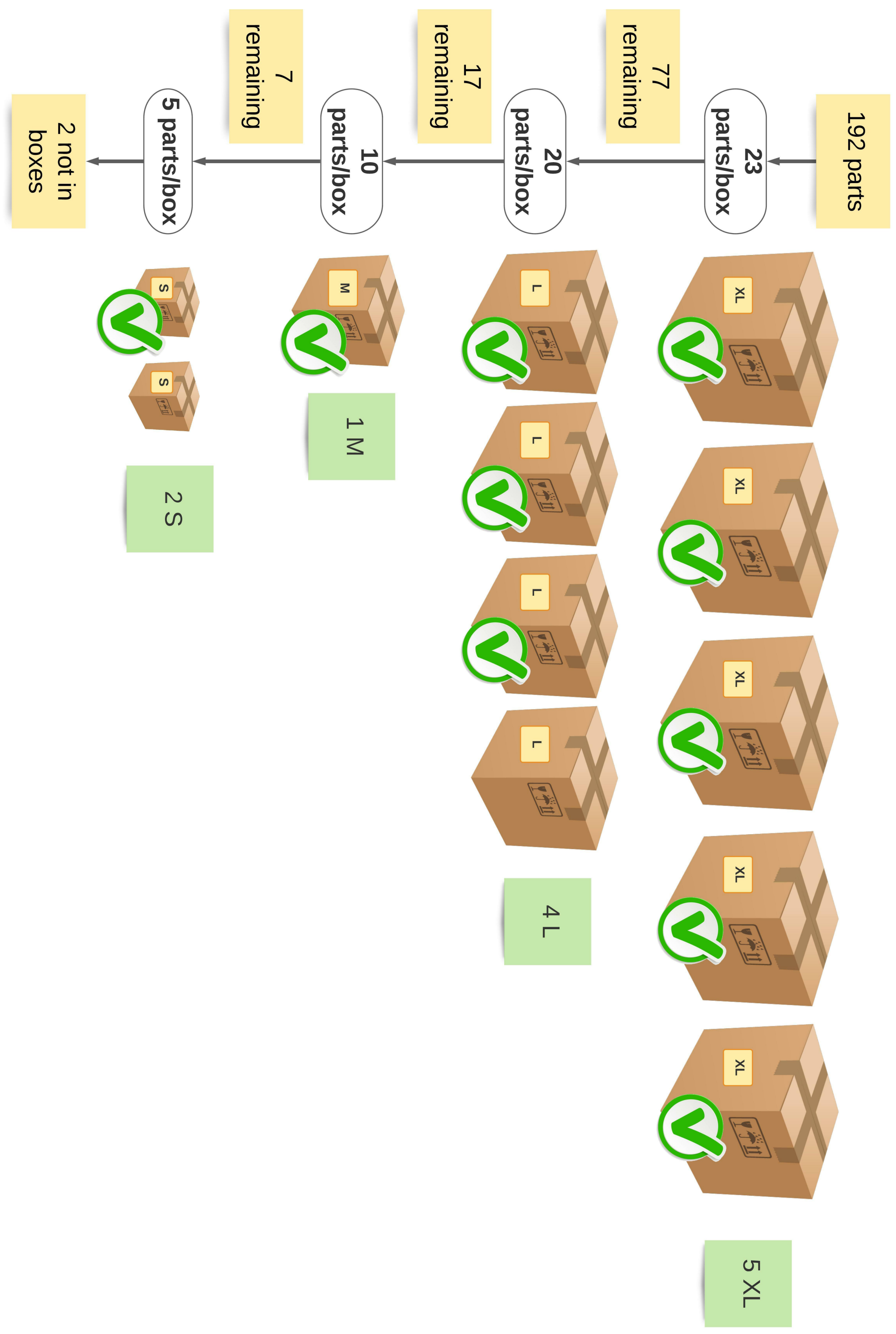

Figure 2: Filling up boxes for 192 parts.

3 Code Structure

Please write your program in one single .cpp file. Below is the structure of this file.

```
/*
function prototypes here
*/

/*
define your data structures here
*/

/*
function definitions here
*/

int main() {
    //call function to get total number of parts
    //call function to get total number of boxes of each type
    //call function to get the max number of parts per box type
    //call function to fill up boxes and to display result
}
```

4 Packaging The Project

1. Questions on assignment can be asked to me (Cc TAs) through emails or on Canvas (Discussions). Do not ask questions to TAs as they will refer to me.
2. Create a project named **Rwa1Section#FirstnameLastname** (where # is your section number). Example: **Rwa1Section1ZeidKootbally**.
3. Write your program.
4. Document your code with Doxygen. There is no need to generate the HTML documentation and there is no need to submit any kind of documentation.
5. Compress your project (zip, rar, etc).
6. Upload it on Canvas.

5 Grading Rubric

- Assignment is 30 pts.

- 20 pts if your program works perfectly. This includes handling bad user inputs.
  - If your program partially works, TAs have the freedom to deduct points as they see fit. I will make sure point deductions are fair.
- 5 pts if you followed the guidelines and best practice. 0 pt for not following at all. 2 pts for partially following.
- 5 pts if you documented your code. 0 pt for not documenting at all. 2 pts for partial documentation.