0.1 Specification of the language to the purpose

To specify the language, so it will make the program as suitable as possible for writing drink machines, we looked at what the central aspects of a drink machine is:

- **A drink:** A drink is central to this machine. A drink should be the product made by the machine, defined by a number of ingredients. A drink should be like a recipe.
- **An ingredient:** An ingredient is the elements of a drink. It will in the machine be contained in a container.
- **A RFID-tag:** A RFID-tag with a drink ID and an amount of how many drinks there are left.
- **A RFID-RW:** A RFID-reader and writer for Arduino, to write and read the content of a RFID-tag.
- **A LCD:** For communicating with the user, a LCD is in most situations preferable.
- **Buttons:** For getting input from users, buttons are a possibility.
- **Mechanism for pouring ingredients:** A mechanism for pouring the right amount of ingredients into the drink.

These are by our assessment the most central aspects of the drink machine system. We will now make a judgment of each of the listed aspects, and see if it is possible to make a structure in the language which will support the programmer on in any other way make it easier implement this aspect in the system.

0.1.1 A drink

The concept of a drink is one of the most central aspects of a drink machine system. A drink should contain a recipe as a list of ingredients, and how much of the ingredients to pour. A drink do also have a name, and should have a form for ID which can be stored on a RFID-tag. We should make a structure which can implement a drink type and assign the recipe to the drink. The structure should be following the same design criterias as the rest of the language, and should be inspirited by the other structures in the language. To fulfill these requirements, the declaration of a drink should have a block with "begin" and "end", and in the block have the "recipe". The result can be seen on listing 1.

```
drink [drinkname] is
begin
  [recipe]
end
```

Listing 1: The first example of a declaration of a drink

We have now decided how to define a element of the type drink. We must now look at the body of the block in the declaration. To make as readable as possible, it should be written a little like a normal recipe. Because of that, we have decided to state it in the form "add [number] of [ingredient]", where [number] is a number representing the amount of the ingredient to add. The declaration of a drink will now be defined as seen on listing 2.

```
drink [drinkname] is
begin
  add [number] of [ingredient];
  add [number] of [ingredient];
  ⋮
  add [number] of [ingredient];
end
```

Listing 2: The final structure of how to declare a drink

In some situations, a drink could be very much alike from an other drink, with only a few changes. For an example could it be a drink with a double shot of alcohol. In this case, it would be the exactly same drink, but with more of one ingredient. It could also be an ingredient that should be remove, for example alcohol to make it non-alcohol ore it someone is allergic to some of the elements in a drink. Because of these situations and many more, it could be preferable to have a way to inherit the recipe from an other drink, and then modify it. The declaration of a drink which inherits from an other drink should be very much alike the normal drinkdeclaration as seen on listing 2, but also with so significant modifications, so it is easy to see that this drink inherits from an other drink. The block structure should be the same, and the way of add an amount of an ingredient should be the same. We have to add a now command to the block statement: "remove". With the remove statement, it should be possible to completely remove an ingredient from a drink. It should be easy to read, and for that we have decided that the declaration statement before the block should be "drink [drinkname] as [drinkname of drink to inherit] but" to say that the drink is as the other drink, but with changes. The final structure will therefore be as seen on listing 3.

```
drink [drinkname] as [drinkname of drink to inherit] but
begin
  add [number] of [ingredient];
  remove [ingredient];
  ⋮
  add [number] of [ingredient];
end
```

Listing 3: The structure of how to declare a drink which inherits the recipe from an other drink

We have now defined how the structure of the type drink should be. To formally define the syntax, we use BNF which can be seen on grammar **??**.

$\langle drinkdcl \rangle \rightarrow$ drink $\langle id \rangle$ is begin $\langle drinkstmts \rangle$ end
  | drink $\langle id \rangle$ as $\langle id \rangle$ but begin $\langle changedrinkstmts \rangle$ end

$\langle drinkstmts \rangle \rightarrow \langle drinkstmt \rangle \langle drinkstmtsend \rangle$

$\langle drinkstmt \rangle \rightarrow$ add $\langle numeric \rangle$ of $\langle id \rangle$

$\langle drinkstmtsend \rangle \rightarrow$; $\langle drinkstmts \rangle$
  | ;

$\langle changedrinkstmts \rangle \rightarrow \langle changedrinkstmt \rangle \langle changedrinkstmtsend \rangle$

$\langle changedrinkstmt \rangle \rightarrow \langle drinkstmt \rangle$
  | remove $\langle id \rangle$

$\langle changedrinkstmtsend \rangle \rightarrow$; $\langle changedrinkstmts \rangle$
  | ;

Grammar 1: The grammar for the drink declaration

0.1.2 An ingredient

With the drink type defined above, we should now focus on, how to define an ingredient. In the psychical machine, an ingredient will in most cases be a liquid in a container. It could also be leaves or fruit, also in a container. When pouring an ingredient, no matter what ingredient it is, the drink machine should in some way (different from the type of ingredient) send a signal to the container to open and pour the wanted amount of the ingredient. When sending the signal in Arduino, you change the output voltage on a pin, so to make it easier for the programmer, we could implement a type container which hold the pinnumber for the signal to the container. The structure of this can be seen on listing 4.

```
container [containername] <-- [containerpin]
```

Listing 4: The structure of how to declare a container

0.1.3 A RFID-tag

For a customer to use the drink machine, they should buy an RFID-tag with information of which drink and how many of the drink they have bought. An solution of these RFID-tags for writing and reading, is to take care of this for the programmer, so they should not deal with how to store the information and read it. It should therefore be provided by functions build into the language. This can be seen in section 0.1.4.

0.1.4 A RFID-RW

To read and write the RFID-tags, an RFID reader and writer is required. To simplify the communication with the RFID-RW, the language could, as mentioned in section 0.1.3.

When writing information to a tag, you should call a function with two parameters: the drink ID and the number of drinks. The function will be called RFIDWrite. A call to the function could be as the example seen on listing 5.

```
call RFIDWrite([drinkID], [Amount]);
```

Listing 5: An example of the call to RFIDWrite.

0.1.5 A LCD

For communicating the the user of the drink machine, a LCD would be preferable like the one seen on figure 0.1.

Figure 0.1: An example of a LCD to use in a drink machine. [?]

Like the function provided to the RFID-RW (see section 0.1.4), it could simplify the programmers job if the language provided a function to print text on the LCD. The function should take two parameters: the string to print on the display and an integer which indicates which line to print the string on. This function will be call LCDPrint. It should also be possible to clear the LCD. The function to do this will be called LCDClear. An example of a call to the functions can be seen on listing 6.

```
call LCDPrint("[text to print]", [linenumber]);
call LCDClear();
```

Listing 6: An example of how to call the LCD functions.

0.1.6 Buttons

To get input from either the staff or the customer, buttons could be used. It could be possible to make a button type in the language, and make some operations of the button easier. This could be making a listener, which will call a function when a button is pressed, or other buttons functions to be build into the language. This is considers as an good idea, but because it is not essensial for the project is will not be implemented, but it could be made if the language was further developed.

0.1.7 Mechanism for pouring ingredients

The mechanism to pour the ingredient would be very different. A liquid will for example not be poured the same way a leaves. For this reason, it has been decided not to supporting the pouring mechanism in any special way in the language, because the possibilities of how the pouring mechanism will be too many.