Assignment 1

This assignment consists of designing and building a set of programs to simulate a building entry controller system and provide a testing harness. The system is similar to that discussed during the lectures – a two-door system with card scanners on each door, a scale that weighs the person after they have entered the space between the two doors, and a human guard that uses a switch to open each of the doors.

Details:

1. People enter the entry control space from the left to enter the building and enter the control space from the right to exit the building.
2. Each person requesting to enter or exit the building scans their unique personal id card containing an integer **person_id** number. The scanner sends a message to the controller with the input information (which door opening is being requested, and which person_id is being used.; (e.g., "left 12345")
3. Only 1 person at a time should be able to be inside the lock.
4. Assume that the door is **self-closing (but not self-locking)**, and that an event (see below under Step 4.a.) will be sent to the controller when the status of the door changes.

Steps:

1. (20%) Design a state machine for the system. Provide the following:
   a. A list of inputs, including persistent saved data (e.g., the weight of the person on entry)
   b. A list of outputs, including persistent saved data.
   c. A list of conditions / events that cause transitions between states.
   d. A list of states
   e. A diagram showing the state machine.

2. (10%) Write a program *des_display* that displays the status of the system – which door is open/closed if there is a user waiting to enter from the left or right, etc. The program should run in the background and print out status information to the console each time a status update is sent to it using a message from the *des_controller* program. The *des_display* program can print out its process id when it first starts up (as in Lab5).

3. (20%) Write a program *des_controller* that runs in the background, operates the state machine for the controller, and directly maintains the persistent data and status for the controller. It should have separate functions for the state handler for each state. Each state handler should perform the actions required for that state, send a message to the *des_display* program to update the display (as required), and then check the exit conditions for that state. When an exit condition is met, the state handler should return the function pointer for the next state handler. The *des_controller* program should print out its process id when it first starts up.

4. (10%) Write a program *des_inputs* that prompts the user for inputs to the controller. This program is simulating all of the input events from the devices; e.g., card readers, door latches, scale.

   Prompt for the input:

   a. The first prompt should be:
      Enter the event type (ls= left scan, rs= right scan, ws= weight scale, lo =left open, ro=right open, lc = left closed, rc = right closed , gru = guard right unlock, grl = guard right lock, gll=guard left lock, glu = guard left unlock)
   b. If the event is the lo, ro, lc, rc, glu, gll, gru, or grl, no further prompt is required.
   c. If the event is ls or rs, prompt for the person_id.
      Enter the person_id
   d. If the event is ws, prompt for the weight.
      Enter the weight

   Once the input is finished, send a message to the *des_controller* program to provide the input "event", and loop back to prompt again.

   This scenario represents a Person entering the building, beginning with a left scan (ls) of the person's ID:

```
ls
12345
glu
lo
ws
123
lc
gll
gru
ro
rc
grl
exit
```

Error Handling

When *des_controller* receives an illegal DES input event from *des_inputs*, then do not transition to an accepting state nor an error state. Rather, the state machine of *des_controller* is to remain in its current state. Only transition to the next accepting state on receiving the expected valid input event from *des_inputs* based on the DES "grammar", as seen in the LeftRightScan of item 3 in the Deliverables section (see below).

Example Startup:

#./des_display &

> The display is running as process_id 77234.
>
> [status update : initial startup]

#./des_controller 77234 &

> The controller is running as process_id 77235.

#./des_inputs 77235

> Enter the device (ls= left scan, rs= right scan, ws= weight scale, lo =left open, ro=right open, lc = left closed, rc = right closed , gru = guard right unlock, grl = guard right lock, gll=guard left lock, glu = guard left unlock)
>
> …..

Steps:

1. Get Lab 5 working! (Use it as a model).
2. Design the state machine.
3. Create a new Momentics workspace named: **cst8244_assign1**
4. Create new projects named **des_display**, **des_controller**, and **des_inputs**. Create these projects as QNX Projects > QNX Executable.
5. Create an include file "des.h" to define the typedefs for structs to be used for sending requests and receiving responses, and for storing the status information.
6. Create the *des_controller, des_display,* and *des_inputs* programs.
7. Test your programs.
8. Format your source code to make it easier for me to read. Momentics IDE will do this for you: Source → Format
9. Verify your projects build cleanly. Please action: a) Project → Clean… and b) Project → Build All

Deliverables

Prepare a zip file that contains the following items:

1. Export your projects as a zip-archive file.
   Momentics IDE provides a wizard to export your project:
   File → Export… → General → Archive File → Next → Select All → Click 'Browse…" →
   Save As: **cst8244_assign1_*yourAlgonquinCollegeUsername*.zip** → Save → Finish

2. The diagram showing your state machine.

3. Upload a zip-file of screenshots showing the run-time behaviour of your assignment. Capture these scenarios:
   a. This scenario, called LeftRightScan:
      ls
      12345
      glu
      lo
      ws
      123
      lc
      gll
      gru
      ro
      rc
      grl
      rs
      54321
      gru
      ro
      ws
      321
      rc
      grl
      glu
      lo
      lc
      gll
      exit

b. This scenario, called ExitScan:
   exit

c. This scenario, called ErrorScan:
   ls
   54321
   lo
   ro
   lo
   ro
   letmeout
   exit

   Notice: "ls 54321" is a valid input event; all other inputs are invalid (i.e. lo, ro, etc.).

Reference Screenshots: your screenshots for the first scenario are to match mine. See "Memo: Reference Screenshots" in the "Assignment 1" module on Brightspace.

4. A "README.txt" file reporting the status of your assignment. Follow this template:

   Title {give your work a title}

   Author
   {Please sign your work. For example @author john.doe@AlgonquinCollege.com
   Include your partner's name (if you have one)}

   {IF you collaborated with a partner, list both names (yours + partners) and provide a brief description of what you worked on and contributed to the assignment.}

   Status
   {Tell me the status of your project. Does your program meet all of the requirements of the specification? Does your program run, and more importantly, does your program behave as expected? Does your program terminate unexpectantly due to a run-time error? Any missing requirements? A small paragraph is sufficient.}

   Known Issues
   {Tell me of any known issues that you've encountered.}

   Expected Grade
   {Tell me your expected grade.}

5. Name your zip file according to your Algonquin College username:
   cst8244_assign1_yourUsername{_yourPartnerUsername}.zip

   For example (partners), cst8244_assign1_bond007_jaws0001.zip
   For example (solo), cst8244_assign1_bond007.zip

6. Upload and submit your zip file to Brightspace before the due date.

   - Collaborated with a partner(s)? <u>Two</u> submissions please; each partner is to submit their own zip-file.