Advanced Programming (I00032)
A DSL to Control an Arduino Microprocessor

Assignment 13

In this assignment you have to make a DSL to lift the abstraction level for microprocessors. Chose the approach to make a DSL from the course that suits your needs the best.

Part of your final mark

To reward your efforts in making this larger assignment it can contribute to your final mark of this course. When your mark for the exam is at least 5, the final mark can be improved by averaging with this exercise. The weight of this exercise will be 25%. The marks will only be averaged if this improves your result. This assignment and assignment 7 can determine together up to 50% of your final mark for this course.

1 Introduction

Microprocessors are small computers with very limited capabilities. The reason to use them are that they are very suited for simple control tasks. For this purpose they are often equipped with special hardware, like sensors (for instance temperature sensors and buttons), and actuators (for instance motors, lamps, and displays).

In this assignment we consider an Arduino microprocessor equipped with a two line LCD display equipped with five buttons. The Arduino is a very popular family of open source microprocessor systems. Documentation can be found on `www.arduino.cc`.

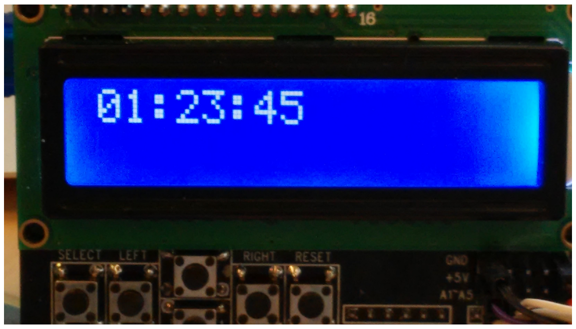

Figure 1: A LCD shield with two lines of 16 characters and 5 user buttons.

The software on an Arduino is written in a dialect of C, see `https://www.arduino.cc/en/Reference/HomePage`. There is a free IDE to support development of Arduino software `https://www.arduino.cc/en/Reference/HomePage`. The Arduino has no operating system. This has the advantage that all hardware, especially the I/O ports, can be controlled directly by the only program executing on this machine. The disadvantage is that there is no support from an operating system.

Each program for the Arduino contains at least the functions `void setup()` and `void loop()`. The `setup` function is executed once at start-up, or after a reset, of the system. After executing the `startup` function the `loop` function is executed repeatedly until the power of the system is disconnected. Each of these functions can have an empty body.

The "Hello world" example for the Arduino blinks the LED connected to pin 13:

```
void setup() {
  pinMode(13, OUTPUT);                // initialize pin 13 as output
}

void loop() {
  digitalWrite(13, HIGH);             // switch LED on (HIGH voltage)
  delay(1000);                        // wait a second
  digitalWrite(13, LOW);              // switch LED off (LOW voltage)
  delay(1000);                        // wait a second
}
```

Since the `delay()` function blocks the single program on the Arduino, this is not the recommended way to write such a program. It is better to look at the number of milliseconds that has passed since the system is switched on and act when this clock indicates that it is time.

```
#define DELAY 1000
boolean ledOn = false;                // status of LED
long lastTime = 0;                    // last status switch

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  if (millis() - DELAY > lastTime) {  // time to change?
    ledOn = not ledOn;
    digitalWrite(LED_BUILTIN, ledOn);
    lastTime += DELAY;
  }
}
```

In this program we can do different things in the loop, even when the system is waiting to switch the status of the LED.

1.1 The `LiquidCrystal` class

One of the most use shields contains a LCD of two lines of 16 characters each. The corresponding library is include by `#include <LiquidCrystal.h>`. A LCD object is made by the constructor `LiquidCrystal lcd(8,9,4,5,6,7)`, see `https://www.arduino.cc/en/Reference/LiquidCrystal` for a complete description. The arguments in this constructor indicate the pins used by this Arduino shield. Various brands of shields use different pins, and even a different number of pins. Hence, we should indicate the pins used in the constructor although it is fixed for a given type of shield. Since there can be several LCD screens connected to a single Arduino, we need to create a real `LiquidCrystal` object here (in contrast with the `Serial` class above). In your DSL you can assume that this LCD with this size and connections is the only relevant shield.

A simple program that shows the message `Hello World!` is:

```
#include <LiquidCrystal.h>                        // add LCD library
LiquidCrystal lcd = LiquidCrystal(8,9,4,5,6,7);   // define lcd object
```

```
void setup() {
  lcd.begin(16, 2);                           // set LCD size
  lcd.print("Hello world!");                  // display message
}

void loop() { }
```

Instead of the given definition of the `LiquidCrystal` object, you will often encounter the statement `LiquidCrystal lcd(8,9,4,5,6,7);` as C++ shorthand for the construction of the object `lcd`.

1.2 Buttons on the LCD-shield

Many LCD-shields are equipped with buttons for user input. Typically there are 6 buttons. The rightmost button is the reset button of the Arduino board and cannot be used in programs.

The other buttons are labelled *Select*, *Left*, *Up*, *Down*, and *Right* (from left to right). To safe I/O pins these buttons are connected to a single analogue input pin: A0. Via a network of resistors, called a voltage ladder, we can determine the button pressed.

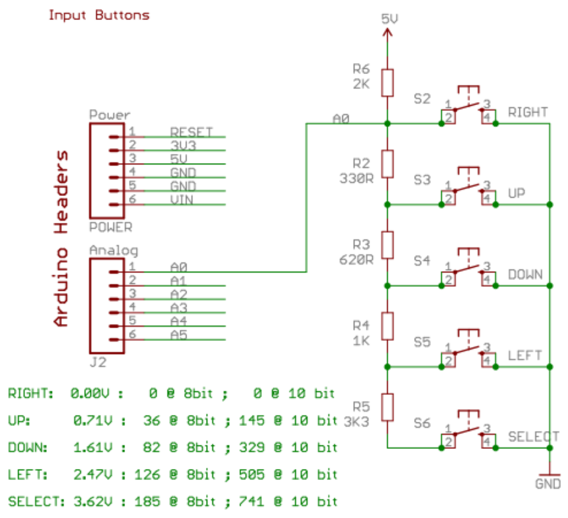
Input Buttons
Arduino Headers
Power
RESET
3V3
5V
GND
GND
VIN
POWER
Analog
A0
A1
A2
A3
A4
A5
J2
5V
R6 2K
R2 330R
R3 620R
R4 1K
R5 3K3
S2
S3
S4
S5
S6
RIGHT
UP
DOWN
LEFT
SELECT
GND
RIGHT: 0.00V : 0 @ 8bit ; 0 @ 10 bit
UP: 0.71V : 36 @ 8bit ; 145 @ 10 bit
DOWN: 1.61V : 82 @ 8bit ; 329 @ 10 bit
LEFT: 2.47V : 126 @ 8bit ; 505 @ 10 bit
SELECT: 3.62V : 185 @ 8bit ; 741 @ 10 bit

Figure 2: The voltage ladder for the buttons of the LCD-shield.

The Arduino Uno R3 used as example has a 10-bit Analog-Digital converter for reading analog inputs. This implies that input voltages between 0 and 5 volt are mapped to number in the range from 0 to 1023. In a perfect world the Analog-Digital-Convertor of pin A0 will produce the numbers listed in Figure 2 when a button is pressed. In the real world there are tolerances for the resistors and inaccuracies in the AD-converter, this result in slightly different readings from input A0 when a button is pressed. A typical program tests a value somewhere between the given numbers to determine the button pressed.

```
#include <LiquidCrystal.h>
#define KEY_COUNT 5

int keyLimits [KEY_COUNT+1]      = {50, 190, 380, 555, 790, 1024};
char keyNames [KEY_COUNT+1] [10]
  = {"Right ", "Up    ", "Down  " , "Left  " , "Select" , "No key"};
```

```
LiquidCrystal lcd = LiquidCrystal(8, 9, 4, 5, 6, 7);

void setup() {
  lcd.begin(16, 2);
}

void loop() {
  int val = analogRead(A0);
  lcd.setCursor(0, 0);
  lcd.print(val);
  lcd.print(" on A0    ");
  for (int i = 0; i ≤ KEY_COUNT; i += 1) {
    if (val < keyLimits[i]) {
      lcd.setCursor(0, 1);
      lcd.print(keyNames[i]);
      break;
    }
  }
  delay(500);
}
```

1.3 Multiple Tasks on a Microprocessor

When we want to execute multiple tasks on a single microprocessor we have to combine them into a single `setup` and `loop` function. Using delays as in the first blink program, these tasks really influences each other. Using the timer as in the second blink program these tasks can run smoothly together, in worst case there is a slight additional delay.

2 Assignment: A DSL for the Arduino with a LCD-shield

Make a DSL to program Arduino microprocessors equipped with the described LCD-shield. You are free to chose one of the implementation techniques for DSL's introduced in the course.

Requirements: your DSL must obey these properties:

1. There must be two different views on programs in your DSL:
   (a) It must be possible to simulate your programs interactively in an iTask program.
   (b) There is a code generator that produces code for the Arduino that is accepted by the IDE. The generated code must look plausible, it is not required to upload your code to a real Arduino and test it. Whenever desired we can plan a session in the New Devices Lab to test your code on a real Arduino.

       Running Clean on an Arduino is no option. There is only 32Kb available to store programs and 2Kb of RAM to store the stack and all variabels in the program. Hence this view must directly produce C-code in the Arduino dialect.
2. Your DSL should provide an abstraction for reading the buttons connected to pin A0. For instance, a function like `isPressed :: Button → Bool` and/or `pressed :: Button` that indicates the status of the buttons[1]. In these definitions `Button` is an enumeration type indicating the buttons of the LCD-shield.

[1]When two or more buttons are pressed simultaneously the hardware ensures only the one with the lowest reading on A0 is detected.

3. It should be possible to run at least two tasks simultaneously: one to read the buttons and one to adjust the display when the clock is running.

4. Programs in your DSL should not block while waiting. This implies that you have to avoid the use of `delay`.

5. There should be at least two test programs for your DSL:

   (a) Score counter for a two person/team game. There are 3 buttons used in this program: one for increasing the score of the first player/team, one for increasing the score of the other team and a reset button. The output on the LCD looks for instance like `"0 - 0"` or `A:` 0; `B:` 0.

   (b) A countdown timer. The start time in minites and seconds can be set by the buttons. The timer can be started, interrupted and reset. Once started the timer counts down from the current time to zero until it is interrupted.

Apart from the requirements for the DSL that must be fulfilled, there are wishes. For a perfect solution these wishes are all fulfilled (as far as possible in an embedded DSL).

- The DSL is strongly typed.
- The DSL provides a task-oriented way of programming.

  Ideally: there can be as many tasks running in parallel as desired; tasks can start other tasks; tasks are parameterized; and task can communicate (by Shared Data Sources and/or task results). This is much more then necessary in the current exercise.
- The Clean compiler checks the variables in the DSL.
- The DSL is a functional programming language.
- The DSL has an appealing syntax.

Hint: Start with one of the DSLs used in the course and adapt it to the requirements of this assignment. Make the initial views very simple. Improve the views by need and as long as time permits.

Making a complete fully feathered task based DSL is a lot of work, this is not necessary in the current exercise.

Deadline

The deadline for this assignment is January 8, 2017, 23:59h.