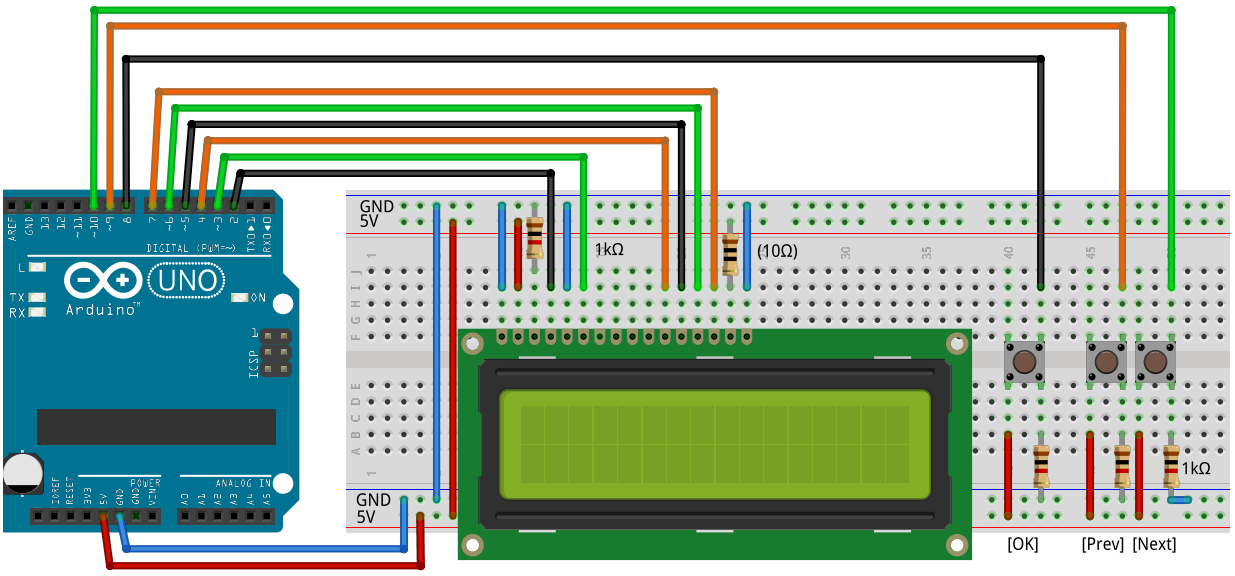
Abgabe

Dieses Übungsblatt ist bis Freitag, 27.06. um 12:00 Uhr per Email an den eigenen Tutoren abzugeben. Benennen Sie die Abgabe bitte eindeutig: "*Matrikelnummer*_Abgabe_*Blattnummer.Format*".

1 Der kürzeste Weg

Das folgende Bild zeigt den Aufbau für ein Navigationssystem mit LCD-Modul und Eingabetastern (*OK* und zwei Taster um zum vorherigen und nächsten Listenelement zu navigieren).



Der in der Vorlesung vorgestellte Dijkstra-Algorithmus sollte bereits ausgeführt worden sein. Den Code dazu finden Sie auf der Vorlesungswebseite. Dabei stehen Ihrem Programm die Daten aus Listing 1 zur Verfügung.

a. Implementieren Sie ein Navigationssystem. Zunächst sollen Sie dabei nur die Strecke von Hamburg nach Saarbrücken anzeigen. Geben Sie für jeden Wegabschnitt die zuletzt passierte Stadt, die zunächst zu passierende Stadt als auch die verbleibende Entfernung bis zum Ziel aus.

   Per Druck auf die entsprechenden beiden Taster soll zum vorherigen oder nächsten Wegabschnitt gewechselt werden können.

b. Wird der nächste Wegabschnitt nach Erreichen des Ziels durch Betätigen des entsprechenden Tasters angefordert, soll die Nachricht "Ziel erreicht. 0 km bis ..." angezeigt werden.

c. **Bonusaufgabe:** Erweitern Sie das Navigationssystem so, dass von einer beliebigen Ausgangsstadt zu einer beliebigen Zielstadt gefahren werden kann. Nutzen Sie für die Auswahl von Herkunft und Ziel dabei die drei Taster.

d. **Bonusaufgabe:** Bei Betätigen des OK-Buttons in einer laufenden Navigation bricht die Navigation nach Rückfrage ab und erlaubt das Starten einer neuen Navigation. Die Rückfrage soll dabei in der ersten Zeile des Displays die Frage "Neue Navigation?" und alle drei Taster zur Auswahl der Antwort ("Ja" oder "Nein") in der zweiten Zeile nutzen.

Listing 1: Eingabedaten für das Navigationssystem

```
const int INF = 100000;
const int CITIES = 6;
const char *names[CITIES] = { "SB", "DO", "F", "HB", "H", "HH" };

const int dist[CITIES][CITIES] = {
  {   0, 354, 177, 593, 532, 692 },
  { 354,   0, 220, 239, 212, 360 },
  { 177, 220,   0, 459, 355, 515 },
  { 593, 239, 459,   0, 130, 121 },
  { 532, 212, 355, 130,   0, 160 },
  { 692, 360, 515, 121, 160,   0 }
};

const int via[CITIES][CITIES] = {
  { 0, 0, 0, 1, 2, 4 },
  { 1, 1, 1, 1, 1, 3 },
  { 2, 2, 2, 1, 2, 4 },
  { 1, 3, 1, 3, 3, 3 },
  { 2, 4, 4, 4, 4, 4 },
  { 2, 3, 4, 5, 5, 5 }
};
```

Beispiellösung.

```
#include <LiquidCrystal.h>

/**
 * Dijkstra
 */

const int CITIES = 6;
const char * name[ CITIES ] = { "SB", "DO", "F", "HB", "H", "HH" };

const int dist[ CITIES ][ CITIES ] = {
  { 0,   354, 177, 593, 532, 692 },
  { 354,   0, 220, 239, 212, 360 },
  { 177, 220,   0, 459, 355, 515 },
  { 593, 239, 459,   0, 130, 121 },
  { 532, 212, 355, 130,   0, 160 },
  { 692, 360, 515, 121, 160,   0 }
};

const int via[ CITIES ][ CITIES ] = {
  { 0, 0, 0, 1, 2, 4 },
  { 1, 1, 1, 1, 1, 3 },
  { 2, 2, 2, 1, 2, 4 },
  { 1, 3, 1, 3, 3, 3 },
  { 2, 4, 4, 4, 4, 4 },
  { 2, 3, 4, 5, 5, 5 }
};
```

```
/**
 * Interface
 */

// Button handling
enum Button { OK = 0, PREV = 1, NEXT = 2 };
int buttons[] = { 8, 9, 10 };
unsigned long lastClicked[3];

/**
  * Blocks until a button is pressed and
  * returns the index of the pressed button
  */
int waitForClick() {
  while (1) {
    for (int b = 0; b < 3; b++) {
      if (digitalRead(buttons[b]) && millis() > lastClicked[b] + 200) {
        lastClicked[b] = millis();
        return b;
      }
    }
  }
}

// LCD-Anzeige
int rsPin = 2, ePin = 3, d4Pin = 4, d5Pin = 5, d6Pin = 6, d7Pin = 7;
LiquidCrystal lcd(rsPin, ePin, d4Pin, d5Pin, d6Pin, d7Pin);

int start, ziel, position;

/**
 * Offers the user the possibility to abort the current navigation
 * - returns true if user chooses to abort
 * - returns false and restores the display content otherwise
 */
bool abortNavigation() {
  //display options
  lcd.clear(); lcd.print("Neue Navigation?");
  lcd.setCursor(0,1); lcd.print("JA NEIN");
  lcd.setCursor(0,1); lcd.cursor();

  bool choice = true; //current choice (y/n)
  //selection mechanism
  while (1) {
    switch (waitForClick()) {
      case OK:
        lcd.noCursor();
        return choice;
      case PREV: case NEXT:
        choice = !choice;
        lcd.setCursor(choice ? 0 : 3, 1);
        break;
    }
  }
}
```

```
/**
 * Displays the current position in the route,
 * thereby correcting overcounting in the position variable.
 */
void printPos() {
  //find current city (thereby correcting overcounting)
  if (position < 0) { position = 0; return; }

  int city = start;
  for (int i = 0; i < position; i++) {
    if (city == ziel) { position = i; return; }
    city = via[ziel][city];
  }
  Serial.print("###");
  Serial.println(name[city]);

  //write position information on screen
  char buffer[100];
  lcd.clear();
  if (city != ziel) {
    sprintf(buffer, "Von␣%2s␣nach␣%2s", name[city], name[via[ziel][city]]);
  } else {
    sprintf(buffer, "Ziel␣erreicht!");
  }

  lcd.print(buffer); lcd.setCursor(0,1);
  sprintf(buffer, "%4d␣km␣bis␣%2s", dist[ziel][city], name[ziel]);
  lcd.print(buffer);
}

/**
 * Interactive mode to click through a navigation route
 */
void navigateRoute() {
  position = 0;
  //navigation interaction
  while(1) {
    printPos();
    switch(waitForClick()) {
    case OK:
      if (abortNavigation())
        return;
      break;
    case PREV:
      position--;
      break;
    case NEXT:
      position++;
      break;
    }
  }
}
```

```
/**
 * Choosing start and aim
 */

/** Serves the user with the message and all cities and allows
  * city selection via buttons.
  * - returns the index of the chosen city
  */
int chooseCity(char msg[]) {
  //print message and options
  lcd.clear();
  lcd.print(msg);
  lcd.setCursor(0,1);
  lcd.print("SB DO F HB H HH");
  int offsets[] = {0,3,6,8,11,13};

  //position cursor on first option
  int choice = 0;
  lcd.setCursor(0,1);
  lcd.cursor();

  //choosing mechanism
  while(1) {
    switch(waitForClick()) {
    case OK:
      lcd.noCursor();
      return choice;
    case PREV:
      if (choice > 0) choice--;
      break;
    case NEXT:
      if (choice < CITIES-1) choice++;
      break;
    }
    lcd.setCursor(offsets[choice], 1);
  }
}

/** Offers the user the possibility to choose start and aim and stores
  * the chosen values in start and ziel.
  */
void chooseRoute(){
  start = chooseCity("Start waehlen:");
  ziel = chooseCity("Ziel waehlen:");
}

/**
 *  Runtime
 */

//we loop between route selection and "navigation"
void loop() {
  chooseRoute();
  navigateRoute();
}
```

```
void setup() {
  //setup lcd display
  pinMode(rsPin, OUTPUT);
  pinMode(ePin,  OUTPUT);
  pinMode(d4Pin, OUTPUT);
  pinMode(d5Pin, OUTPUT);
  pinMode(d6Pin, OUTPUT);
  pinMode(d7Pin, OUTPUT);

  lcd.begin(16, 2);
  lcd.noCursor();

  //setup buttons
  for(int i = 0; i<3; i++){
    pinMode(buttons[i], INPUT);
    lastClicked[i] = millis();
  }

  Serial.begin(9600);
}
```

2 Felder

Realisieren Sie die folgenden Funktionen mithilfe von Feldern. Achten Sie dabei auf die korrekte Behandlung von Fehlern und Randfällen. Die Funktion `printAry()` wurde im letzten Übungsblatt beschrieben und kann wiederverwendet werden.

a. Entwickeln Sie eine Funktion `void arrayCopy(int *fromAry, int *toAry, int numElements)`, die `numElements` Elemente von `fromAry` nach `toAry` kopiert. Gehen Sie dabei davon aus, dass die Felder ausreichend viele Elemente enthalten.

```
int a[] = {3, 1, 2, 5, 8, 4};
int b[] = {0, 0, 0, 0, 0, 0};
arrayCopy(a, b, 3);  printAry(b, 6);  // => [3, 1, 2, 0, 0, 0]
arrayCopy(a, b, 6);  printAry(b, 6);  // => [3, 1, 2, 5, 8, 4]
```

b. Erweitern Sie die Funktion zu `void arrayCopyPos(int *fromAry, int fromStartPos, int *toAry, int toStartPos, int numElements)`, die `numElements` Elemente beginnend von Position `fromStartPos` aus `fromAry` nach `toAry` kopiert, beginnend von `toStartPos`.

```
int a[] = {3, 1, 2, 5, 8, 4};
int b[] = {0, 0, 0, 0, 0, 0};
arrayCopy(a, 0, b, 3, 3);  printAry(b, 6);  // => [0, 0, 0, 3, 1, 2]
arrayCopy(a, 3, b, 0, 2);  printAry(b, 6);  // => [5, 8, 0, 3, 1, 2]
```

Beispiellösung.

```
void arrayCopy(int *fromAry, int *toAry, int numElements) {
    for (int i = 0; i < numElements; ++i) {
        toAry[i] = fromAry[i];
    }
}

void arrayCopyPos(int *fromAry, int fromStartPos, int *toAry, int toStartPos,
    int numElements) {
    arrayCopy(fromAry + fromStartPos, toAry + toStartPos, numElements);
}
```

3 Zeiger

Realisieren Sie die folgenden Funktionen mithilfe von Zeigern. Achten Sie dabei auf die korrekte Behandlung von Fehlern und Randfällen. Die Funktion texttttprintAry() wurde im letzten Übungsblatt beschrieben und kann wiederverwendet werden.

a. Implementieren Sie die Funktion `void inlineSort2(int *min, int *max)`. Es werden Zeiger auf zwei Zahlen als Parameter übergeben. Nach Aufruf soll die kleinere der beiden Zahlen in `min` sein und die größere der zwei Zahlen in `max`.

```
int a = 5; int b = 6;
int c = 3; int d = 2;

inlineSort2(&a, &b);
Serial.println(a); // => 5
Serial.println(b); // => 6

inlineSort2(&c, &d);
Serial.println(c); // => 2
Serial.println(d); // => 3

inlineSort2(&b, &d);
Serial.println(b); // => 3
Serial.println(d); // => 6
```

b. Erweitern Sie die Funktion auf drei Parameter. `void inlineSort3(int *min, int *mid, int *max)` erhält drei Zahlen und sorgt dafür, dass die kleinste der Zahlen nach dem Aufruf in `min` enthalten ist, die größte der Zahlen in `max` und die verbleibende Zahl in `mid`.

```
int a = 5; int b = 6; int c = 3;
int x[] = {3, 1, 2, 5, 8, 4};

inlineSort3(&a, &b, &c);
Serial.println(a); // => 3
Serial.println(b); // => 5
Serial.println(c); // => 6

inlineSort3(&c, &b, &a);
Serial.println(a); // => 6
Serial.println(b); // => 5
Serial.println(c); // => 3

inlineSort3(&x[0], &x[1], &x[2]);
printAry(x, 6);  // => [1, 2, 3, 5, 8, 4]

inlineSort3(&x[3], &x[4], &x[5]);
printAry(x, 6);  // => [1, 2, 3, 4, 5, 8]
```

Beispiellösung.

```
void inlineSort2(int *min, int *max) {
    if (*min > * max) {
        int temp = *min;
        *min = *max;
        *max = temp;
    }
}

void inlineSort3(int *min, int *mid, int *max) {
    inlineSort2(min, mid);
    inlineSort2(mid, max);
    inlineSort2(min, mid);
}
```