C-Basics

This appendix provides some basics for programming in C, which should give you the knowledge to program the hybrid controller in C.
This is by no means an exhaustive introduction to C programming and will provide you only with the information needed to complete the homework and understand how the arduino code for the homework works.

Basic Operations

This section explains some basic operations in C.

- **Declaration and Initialization of Variables**
  In C you need to declare your variables first and you can initialize them then by giving them a specific value. Lets say we have a variable x, then this would look like

  ```
  DataType x; // declaration
  x = someValue; // intialization
  // or both together
  DataType x = someValue;
  ```

  The data type defines which values can be stored in the variable. Data types used in the Arduino code are described the next section.

- **Arithmetic operators, comparison/relational operators and logical operators**
  As in Matlab you can use the basic arithemtic operators for addition(+), subtraction(-) multiplication (*) and division (/).
  To compare/relate two variables the following operators are available

  ```
  // checks for equality of a and b
  a == b
  // checks for inequality of a and b
  a!=b
  // checks if a is greater (or equal) than b
  a>b (a>=b)
  // checks if a is less (or equal) than b
  a<b (a<=b)
  ```

  The logical operators in C are

  ```
  a&&b //logical AND
  a||b //logical OR
  !a   //logical negation
  ```

  Instead of the boolean data type you can use integer variables for logical operations in C. Everything non-zero means True and zero means False.

- **Arrays**
  The use of arrays is slightly different than in Matlab. You can declare a variable as an array of a specific data type as follows

  ```
  DataType x[ArraySize];
  ```

ArraySize indicates the size of the array. In contrast to Matlab the array is indexed from 0 to *ArraySize*-1 and you can get the array values with

```
x[0] // first value in array
x[5] // sixth value in array
x[ArraySize-1] // last value in array
```

If you neglect the array size in the declaration and initialize it directly, the compiler will create an array of the size which fits exatly the values you put in to the array.

```
//creates an int array of size 5
int x[]={1,2,3,4,5};
// does the same
int x[5]={1,2,3,4,5};
```

- **Comments**
  Comments can be useful to describe your code. There are two types of comments.
  The first type is //. It comments everything after it in the same line.

  ```
  int x = 5; // integer x gets the value 5
  ```

  The second type is /* code to be commented */. It comments everything after /* until it reaches */.

  ```
  /* This comments
     three lines
     now */
  ```

As you might have recognized in the text above, **every line/statement** you write in C **has to be ended with a semicolon** ;.

Data Types for Variables

Variables in C need a data type. Those data types have to be defined when the variable is declared. The following data types might be of use in this homework:

- **int**
  It declares the variable to be an integer. This means only integer values can be stored in this variable. Beware that if you assign a value like

  ```
  int x = 0.5
  ```

  the value saved in x will be 0, since 0.5 is not an integer. So only declare variables as an integer when you know that it can only have integer values!

- **double**
  It declares the variable to be a floating point number.

- **char**
  This declares the variable to be a character. Characters have to be indicated by 'characterYouWantToStoreInTheVariable'. So

  ```
  char x='a';
  ```

would give the variable x the value of *a*. Furthermore strings can be declared by using a character array

```
char x[charArraySize] = "This␣is␣a␣string";
```

charArraySize indicates how many characters can fit in the string.

One more useful data type is an enumeration, which can be used to declare your own data types. The syntax is

```
enum typeName { attribute1, attribute2,...,attributeN
    };
```

where *typeName* and the attributes can be chosen by the user. Variables with this new data type are for example declared and initialized with

```
enum fruitsInGerman { Apfel, Pfirsich, Banane};
enum fruitsInGerman apple; // declaration
apple = Apfel; // initialization
```

Furthermore enumerations can be used in switch-case statements. An example for that can be seen in the provided Arduino code, where the buffer state is saved in an enumeration and a switch case statement is used to determine what to do in which buffer state.
Another thing worth mentioning is the type classifier *const*, which is used in the provided Arduino code. It declares a variable to be constant and the variable has to be initialized directly.

```
const int x = 5; // The value of x is fixed to 5
```

It is also possible to cast the content of a variable into another data type.
An example for that is

```
double x = 5.687;
int y = (int) x; // y gets the value 5
int y = (int) 5.687; // this gives the same result
```

So by using the prefix (dataTypeToCastTo) you can cast one data type into another.

If-conditions, switch-case statements and loops

- **if-else-condition**
  The syntax for an if-else condition is

  ```
  if (statement)
  {
          //if statement is true do something
  } else
  {
          //otherwise do something else
  }
  ```

  Directly after the *else*-statement could also follow another *if*-statement or the *else*-statement could also be neglected, if you don't want to do something else for a false statement.

- **switch-case statements**
  The syntax for a switch-case statement is

```
switch(variable) {
case someCaseOfTheVariable:
        //do something for this case
        break;
case someOtherCaseOfTheVariable:
        //do something for this case
        break;
        .
        .
        .
case lastCaseOfTheVariable:
        //do something for this case
        break;
default:
        // if no case fits the variable
        break;
}
```

  variable determines the variable you want to check the cases for. The case statements represent possible values for the variable, while the statement jumps into the default case, if the variable has a value not matching any of the cases. The default statement could also be omitted.

- **while-loop**
  An example syntax of a while-loop is given by

```
while(statement)
{
        //do something every iteration
}
```

  This loop is executed as long as *statement* is true.

Handy functions in c

- **sin(x)** and **cos(x)**
  sin(x) and cos(x) calculate the sine och cosine of x, where x is in *radians*! If you have a variable y, which is given in degree, you can transform it into radians using the following code in your C program

```
x = y*PI/180
```

- **atan2(y,x)**
  atan2(y,x) computes the angle between a vector [x,y] and the origin. The output lies in the interval $[-\pi, \pi]$. An advantage of atan2(y,x) is that it considers the signs of x and y to determine the quadrant the coordinates are in.

- **abs(x)**
  abs(x) returns the absolute value of x.

- **Serial.print(...)**
  Since the Arduino has no console, you have to use the Serial port to send messages, which will be printed out in the GUI. Those messages are a good way to check upon the controller state or send some debug values in order to see if the controller is doing the right thing.
  The function to print/send message over the serial port is Serial.print(...). There are three different version of this function, which you can use.
  The first version is

  ```
  Serial.print("Message␣you␣want␣to␣send");
  ```

  Here the input is just a string. The second version is

  ```
  Serial.print(int i,int base);
  ```

  This can be used to print out the value of an integer variable. The second variable is for choosing a base/format for the variable. The base can be BIN or 2 for base 2, OCT or 8 for base 8, DEC or 10 for base 10 and HEX or 16 for base 16.[1] If you neglect the base parameter it is automatically set to a base of 10, which is in our case the only reasonable choice.
  The third version is

  ```
  Serial.print(double d, int round)
  ```

  The third implementation is used to print out a value of a double variable. The second variable defines how many digits after the comma, you want to use. The double variable is rounded according to this. If you neglect the second variable it is automatically chosen to 2.
  A syntax example is

  ```
  int i=5;
  double d=3.45673
  Serial.print("The␣integer␣value␣is␣"):
  Serial.print(i,DEC);
  Serial.print("␣and␣the␣floating␣point␣value␣is␣");
  Serial.print(d,3);
  Serial.print(".\n");
  ```

  This gives the output

  ```
  The integer value is 5 and the floating point
     value is 3.457.
  ```

  The last call or Serial.print introduces a helpful replacement character. "\n" is the newline character, which ends the current line, so that the next message is printed in a new line.

[1] The simulation does not support the base of 2