# Assignment 1:
# Intro to C programming and FreeRTOS

1DT056: Programming Embedded Systems
Uppsala University

January 19th, 2011

## Installation of the development environment

In the exercises and the first lab of this course, we are going to use the MDK-ARM developer kit and the $\mu$Vision IDE for implementing embedded software. Both tools are commercial products by ARM for developing embedded systems on the basis of ARM micro-controllers and other processors. In the scope of the course, we are going to use evaluation versions of the tools that are available free of charge; those versions offer only a restricted feature set compared to the full versions, but are sufficient for our purposes.

To install this software on your own computer or on computers in the lab room 1313, you need to download it from the web page `https://www.keil.com/arm/demo/eval/arm.htm`. The tools are native Windows applications, but can be used without problems on Linux computers with the help of Wine.

After the installation of MDK-ARM and $\mu$Vision, download the development project `http://www.it.uu.se/edu/course/homepage/pins/vt11/lab_env.zip` that can be used as starting point in all of the following exercise questions. Unpack the archive and open the project `lab_env` using $\mu$Vision. This project contains definitions and firmware for the STM32F103 processor (an ARM CORTEX M3 controller), as well as the FreeRTOS operating system.

To write, compile, and simulate/debug your own code, you can start by modifying the source file `main.c` of the `lab_env` project.

Further information is available in the following places:

- Description of the STM32F103 processor:
  `http://www.keil.com/dd/chip/4794.htm`
- Documentation of the libraries provided by MDK-ARM:
  `http://infocenter.arm.com/help/topic/com.arm.doc.dui0475b/index.html`

- FreeRTOS API: `http://www.freertos.org/a00106.html`
- Books on FreeRTOS (we recommend the Generic Cortex M3 edition): `http://www.freertos.org/Documentation/FreeRTOS-documentation-and-book.html`

# Exercises

## 1. C Programming with arrays

Write the function threeColorsSort that takes as input an array of integers in the range of 0 and 2 (0, 1 and 2 only), and arrange them in an increasing order:

```
void threeColorsSort(int * theArray, int arraySize)
```

## 2. C Programming with strings

In this exercise, you will practice how to program with pointers and strings. Without using any library functions, write a C function

```
void append(char* str1, char* str2) { ... }
```

that takes as argument two strings `str1`, `str2` and appends `str2` to `str1`. After calling `append`, the pointer `str1` is supposed to point to the concatenation of (the original) `str1` and `str2`. The caller of `append` has to make sure that enough memory for the result of concatenation is available at the memory address that `str1` points to.

### Example

```
char x[12] = { 'H', 'e', 'l', 'l', 'o', ' ',
               0, 1, 2, 3, 4, 5 };
char *y = "world";
append(x, y);
// now "x" contains the string "Hello world"
```

Your implementation needs to make sure that the output string (pointed to by `str1`) remains a well-formed string. Recall that, by definition, a string in C is an array of characters terminated with zero.

Is it possible that an invocation of `append` changes the string that `str2` points to? Argue why this is not possible, or give an example program where this happens. In the latter case, make sure that your implementation of `append` behaves in an acceptable manner also in such situations (e.g, your program is not supposed to end up in an infinite loop).

## 3. C Programming with function pointers

The general understanding of a pointer is the memory address of some kind of data (integer, array, string, structure, etc ...).
A pointer can be as well pointing to a function.

**Example** Imagine we need to perform several kind of arithmetic operations on integers, let say: multiplying by two, resetting to zero, inverting the integer sign, etc ...
Therefor we define the following functions:

```
void double(int * a) {...}
void reset(int * a) {...}
void invert(int * a) {...}
```

Example of use of one of those functions:

```
int a;
a = 5;
double(&a);
/* (a == 10) is TRUE */
```

We can now define a general function pattern using function pointer definition:

```
void (* arithmeticFuncPtr) (int *);
```

Notice that the star is related to the function not to the returned value. If we consider an integer returning function, we would have the following:

```
int (* funcPtr) (int );
/* funcPtr is a pointer to a function that takes as argument
    an integer an returns an integer.*/

(int *) funcPtr (int );
/* funcPtr is a function that takes as argument
    an integer an returns a pointer to an integer.*/

(int *) (* funcPtr) (int );
/* funcPtr is a pointer to a function that takes as argument
    an integer an returns a pointer to an integer.*/
```

Use of the previously defined arithmetic pointer:

```
int a = 5;

/* Assign the double function address to the function pointer.*/
```

```
arithmeticFuncPtr = &double;

/* Call of the double function using the arithmeticFuncPtr*/
(*arithmeticFuncPtr)(&a);

/* (a == 10) is TRUE */
```

In fact, thanks to their power, function pointers are used quite often in embedded systems C programming.

**Here is what you need to do:**

1. Write the arithmetic functions double, reset and invert.
2. Write the function applyTo that takes as input a function pointer "func", an array of integers "tab" and the array size "size", and applies the pointed function to all the elements of the array:

   ```
   void applyTo(void (* func)(int *), int * tab, int size)
   ```

3. Use applyTo function to double the content of a 10 integers array .

## 4. Debugging

This exercise is about a bug that occurred in a real-world embedded system. In this system, a C function was used to compute the current year, given the (positive) number of days passed since December 31st 1979:

```
int days2years(int days) {
  int year = 1980;

  while (days > 365) {
    if (isLeapYear(year)) {
      if (days > 366) {
        days -= 366;
        year += 1;
      }
    } else {
      days -= 365;
      year += 1;
    }
  }

  return year;
}
```

The function `isLeapYear` used in the body returns 1 if `year` is a leap year, 0 otherwise.

For instance,

$$\texttt{days2years}(1) = 1980$$
$$\texttt{days2years}(400) = 1981$$

Analyse the implementation of `days2years`, and find situations in which the function behaves erroneously. Explain why the kind of bug observed here is hard to detect, both theoretically and practically, and why it is particularly critical for embedded systems.

### 5. Programming with FreeRTOS tasks

Starting from the project skeleton provided in `http://www.it.uu.se/edu/course/homepage/pins/vt11/lab_env.zip`, write a FreeRTOS program that continuously (and at the correct points in time) writes seconds and tenths of seconds since controller start-up to the USART 1 device:

```
0s
0.1s
0.2s
0.3s
...
1s
1.1s
...
```

To this end, you have to change the `main` function in the project to spawn a time-triggered task that periodically takes care of generating the output. Use the function `vTaskDelayUntil` to wait for the correct amount of time before generating the next time stamp and sending it to the serial device. Strings can be sent to USART 1 using the library function `printf`, which in the `lab_env` project has been redirected to this device.

## Submission

Solutions to this assignment are to be submitted before the lecture on

**Wednesday, January 26th, 2011, 13:15, room 1245.**

No solutions will be accepted after the lecture.