Concordia University
Department of Computer Science and Software Engineering

SOEN422 Fall 2021

—

Lab 2
Bare-Metal and Embedded Software Components in C/C++

## Purpose

This lab aims to access input/output ports using appropriate registers and to apply the evolution of the predominant paradigms of software development from the procedural to the object-based and object-oriented approach[1]. For this lab, students are to present a development progression starting with a blink LED sketch on Arduino, and after that, the same blink LED application using (1) a corresponding bare-metal C programming version, (2) a concrete data structure (CDS), (3) an abstract data structure (ADS), and (4) an abstract data type (ADT). Finally, three tasks (5,6, and 7) are to implement a proper data structure in embedded systems: an ADT queue in C, a C++ wrapper, and a complete rewrite in C++.

### Task #1: Compiling and Running a Blink LED program in C using I/O port registers

Compile, upload, and run the Blink LED bare-metal program in C (as presented in Lecture 3)[2]

### Task #2: Implementing a Blink LED as a CDS

```
// LedCDS.h - CDS for an LED

#define LedDigitalPin   13

typedef struct {
    int  digitalOutPin;
    bool status;
} Led;

Led led;

void Led_Init(void) {
    // Your code ...
}

void Led_TurnOff(void) {
    // Your code ...
}
```

[1] Presented during a lecture on "Small and Reusable Object-Based Data Types in C for Embedded Systems.[1]"
[2] Both `bmv_blinkLed.bat` and `bmv_BlinkLed.c` files are available on Moodle in the Lab2 folder.

```
void Led_TurnOn(void) {
    // Your code ...
}
```

The test program:

```
// TestLedCDS.c - Test CDS for an Led

#include "LedCDS.h"

void main() {
    Led_Init();            // setup()

    while (true) {         // loop()
        // Your code ...
    }
}
```

Give the main advantage and the main disadvantage for implementing a LED Blink as a CDS.
Main advantage?
Main disadvantage?

## Task #3: Implementing a Blink LED as an ADS

Do an implementation of a Blink LED as an ADS.

Give the main advantage and the main disadvantage for implementing a LED Blink as an ADS.
Main advantage?
Main disadvantage?

## Task #4: Implementing a Blink LED as an ADT

```
// LedAdt.h - ADT Led interface

#ifndef LedAdt_h
#define LedAdt_h

#include "udef.h"

        struct LedDesc;      // Forward ref
typedef struct LedDesc* Led; // Opaque type

public Led   Led_New(void);
public void  Led_TurnOff(Led this);
public void  Led_TurnOn(Led this);
public void  Led_Delete(Led this);

#endif /* LedAdt_h */
```

```
/* LedAdt.c - ADT Led Implementation */
```

```
#include <stdlib.h>    // malloc, free
#include "LedAdt.h"

#define LedDigitalPin   13

typedef struct LedDesc {
    int  digitalOutPin;
    bool isOn;
} LedDesc, *Led;

#define LedDigitalPin   13

typedef struct {
    int  digitalOutPin;
    bool isOn;
} Led;

Led led;

private void Led_Init(Led this) {
    // Your code ...
}
public void Led_TurnOff(Led this) {
    // Your code ...
}
public void Led_TurnOn(Led this) {
    // Your code ...
}
public Led Led_New(void) {
    // Your code ...
}
public void Led_Delete(Led this) {
    // Your code ...
}
```

The test program:

```
// TestLedADT.c - Test ADT for an Led

#include <stdio.h>

#include "LedAdt.h"

void main(void) {
    Led led = Led_New();   // setup()

    while (true) {         // loop()
        // Your code ...
    }
}
```

In the previous example, find the main advantage and the main disadvantage for implementing a Bling LED as an ADT.
Main advantage?
Main disadvantage?

## Task #5: Implementing a Queue of Integers as an ADT in C

Based on the following `Queue.h` header file, implement an ADT for a queue of integers and write a program to test it.

The `udef.h` header file (below) contains applicable user definitions for your labs. The use of the purposes of `private` and `public` brings greater clarity and more adequately emulates the principle of encapsulation provided by the modular and base-object languages that do not exist in C.

```
// Queue.h - ADT Queue of integers interface

#ifndef Queue_h
#define Queue_h

        struct QueueDesc;        // Forward ref
typedef struct QueueDesc* Queue; // Opaque type

public Queue Queue_New(void);
public void  Queue_Add(Queue this, int value);
public int   Queue_Remove(Queue this);
public bool  Queue_IsEmpty(Queue this);
public void  Queue_Delete(Queue this);

#endif /* Queue_h */
```

## Header File `udef.h`

```
/* udef.h -- User Definitions
//
// Copyright (C) 1999-2021 by Michel de Champlain
// All rights reserved.
//
*/

#ifndef __udef_h
#define __udef_h

#define private            static
#define public          /* public */

#endif // __udef_h
```

## Task #6: Implementing a Queue of Integers as an ADT Using a C++ Wrapper

Based on Task #5, implement an ADT using a C++ wrapper and write a program to test it. See section 9.1 in the paper [1] presented in Lecture 3.

## Task #7: Implementing a Queue of Integers in C++

Rewrite a complete implementation in C++ (only) and write a program to test it. See section 10 in the paper [1] presented in Lecture 3.

## References

[1] M. de Champlain and B. G. Patrick, Small and Reusable Object-Based Data Types in C for Embedded Systems, Embedded Systems Conference, Silicon Valley (2009), Updated and Adapted on September 12, 2020.

## Individual Lab Report Submission

Each student is expected to create a `.zip` file that he will upload to Moodle for grading. The following is expected in the archive:

- A folder containing your source code
- A report in **.PDF** format, no other formats will be accepted

The individual lab report is expected to contain the following:

1. An introduction containing a problem statement, as well as abbreviations and acronyms used.
2. Physical resources used to complete the lab: This includes hardware components used and how they may be configured.
3. Software resources used to complete the lab: This section details any software used including IDEs, compilers, loaders, specific configurations and procedures needed to complete the labs.
4. Program snippets: This section contains snippets from your code related to each task. Your code must be commented by explaining the logic behind it. The code for each task should be in separate source files.
5. Reference Code Common: Discuss all the libraries used in your code and explain briefly how and why they were used.
6. Discussion and Conclusion.

**Soen422_Lab2_Lastname_StudentID.zip (6%) - Due 11:59 PM - Tuesday, October 5, 2021**