Soen 422

Dr Michel de Champlain

Lab # 2

Jason Klimock
26322298

1 Physical Resources

- Arduino Nano
- Laptop
- Usb-c cable

2 Software resources

- IDE: Vim, CLion
- avr-gcc for task 1-4
- gcc for task 5-7

3 Referance Code Common

- avr/io.h : avr libray for C, used for pin access
- util/delay.h : used for delay function, based on cpu
- stdbool.h : Boolean value for C
- limits.h : minimum int value as constant

4 Problem statments and discussion

4.1 task 1

I compiled and ran the professors file using the follwoing make file. I then used this make file for the rest of the arduino sections. The command were all on one line though, but other wise they wont fit in this report. Code worked without issues.

```
default:
        avr-gcc -Wall -DF_CPU=16000000UL -mmcu=atmega328p -o TestLedADS.bin
        TestLedADS.c LedADS.c
        avr-objcopy -O ihex TestLedADS.bin TestLedADS.hex
        sudo avrdude -c arduino -p m328p -P /dev/ttyUSB0 -b 57600 -D -U
        flash:w:TestLedADS.hex:i
```

4.2 task 2

Task 2 was to implement blink as a concrete data structure (CDS). Nothing in this code is private, but it does mimic an object oriented approach. However another down side is that

we can only have one instance of the data structure. We define the struct that contains the Led information and the modifing function in the same head file so everthring is visable. The all we need to do is call these function in a implementation file. But this exposes everything to a user and violated the principle of data hiding.

```
// define the led as a struct
typedef struct {
        int digitalOutPin; // to set digitalOutPin
        bool status; // flag for LED status, ture for on, false for off
} Led;

Led led; // declare instance of Led

// constructor for led
void Led_init(void) {
        // set the figital pin number on int the Led struct to the in built
        //led value
        led.digitalOutPin = LedDigitalPin;
         //set the DDRB as the built in LED
        DDRB |= (1 << (led.digitalOutPin));}
```

As an example here, wwe can see all the port and implementation details in this header file.

4.3 task 3

Task 3 is to implement blink as an Abstract Data Structure(ADS). An ADS has an advatange ofver a CDS in that is hides the data from the user by providing function interfaces only in the header file and the function implementations in the .C file. This effectivly makes the functions public and the struct date private. This also allows us to change the implementation without influencing any use cases of the ADS. The main down side for this is in a lack of reusability: We can only have one instance per implementation file.

4.4 task 4

Task 4 is to implement blink as an Abstract Data Type (ADT). The ADT is sligtly less efficent in terms of both time and space, when compared to the two previous program structures. It used dynamic allocation (malloc), which places the object on the heap which is less space efficent.The functions pass and extra variable of the ADT. This has two effects, first it makes data access slightly less efficent. Second, it allows us to have multiple instances of a ADT in an implementation. This is useful for data structures such as a queue, that we make in parts 5-7. It also has all the data hiding features of the ADS. The many instances allow us to treat it as a data type that can be added to any program.

4.5 task 5

Task 5 asked us to move away from the arduino and to create a queue of integers as an ADT. We have the data hiding present in the header file by providing only function interfaces and struct foward declarations. As seen here, note that an instance of Queue is passed to each function so that many instances of Queue maybe used in a program:

```
#ifndef Queue_h
#define Queue_h
#include <stdbool.h>

struct QueueDesc; // Foward declaration
typedef struct QueueDesc* Queue; // type def of opaque type
// function declarations
Queue Queue_New(void);
void Queue_Init(Queue _this);
void Queue_Add(Queue _this, int value);
int Queue_remove(Queue _this);
bool Queue_IsEmpty(Queue _this);
void Queue_Delete(Queue _this);

#endif
```

Since this is a fixed length Queue I chose to implement this as a circular queue, for linear insertion and removal of elelments.

4.6 task 6

Task 6 asked us to implement a Queue ADT using a C++ wrapper. For this task, I reused the same header and implementation file as in task 5. The c++ wrapper, allows us to define a c++ class and functions with inline calls to functions in C. This seems to mimic the adapter design patter, where the C++ is the adapter. the Extern keyword in the follwoing header tell the compiler to use external linkage and the C identifies this as C code. it is also worth noting that we can apply public and private modifier to our old C implementation using the wrapper. This is advantageous for data hiding.

```
// Queue.hpp C++ interface of a C queue
#ifndef __Queue_hpp
#define __Queue_hpp
// including C header file
extern "C" {
#include "Queue.h"
}
```

```
namespace Sys { // defintion of namespace to not clash with std Queue
// wrapper class defintion
    class Queue {
    private:
        QueueDesc* _queue; // struct defined as private
    public:

// constructor in line defintion of call to C Queue struct function

    Queue(){_queue = Queue_New();}

// destructor in line defintion of call to C Queue struct function

                ~Queue(){Queue_Delete(_queue);}

// Adds value to queue, in line defintion of call to C Queue struct function

    void Add(int value){Queue_Add(_queue, value);}

// removes value from queue, inclince call to C struct function ‘

                int Remove(){return Queue_remove(_queue);}

// return empty status, call to C struct

                bool IsEmpty(){return Queue_IsEmpty(_queue);}     };
};

#endif
```

4.7 task 7

Task 7 asked us to implement the integer Queue in pure C++. This includes all the benifits of OOP and like the previous two task I also implemented this as a circular queue. I inlined the private member variables and defined them as constant becuase they will never chage. Out of all the Q implementations this is the cleanes to use and implement, becuase we can pass fewer variable and have an object. We can also take advantage of static memory allocation, which was not possible in the C implementation.

5 Conclusion

These tasks helped to develop a better understanding of the different way we can define data strcutres and programs in embedded systems. For task 5-7 I used aunit to generate tests and found that tests will not work with more than 9 lines per test. I think this is a bug. I was able to use the same test file for task 6 and 7 since the interface of the Queue class was the same. This illustraits the advantage of using C++ wrappers: if we have an existing C class, we can easily turn it into a C++ class withthe wrapper.