Embedded Systems Programming
Assignment 3.2

Test Driven Development

Steinarr Hrafn Höskuldsson

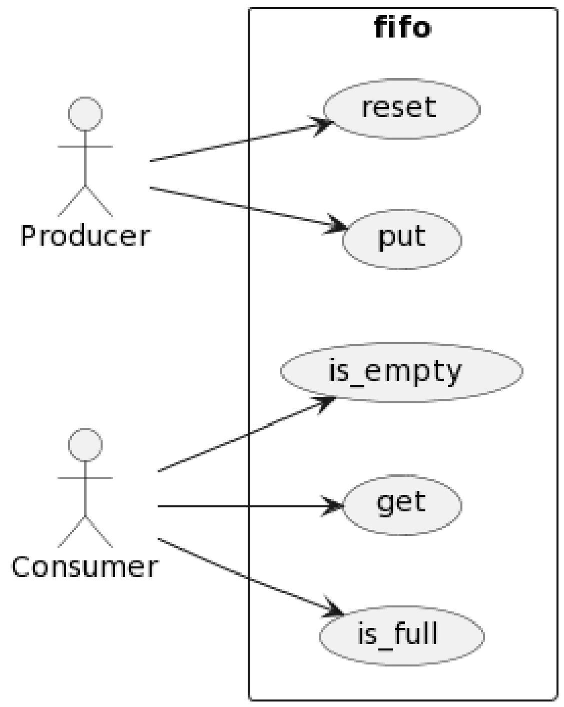
September 20, 2022

Part 1

A new project was created in PlatformIO and a use case diagram using plantUML was written.

```
@startuml "Use case of FIFO Queue"
left to right direction
actor "Producer" as producer
actor "Consumer" as consumer
rectangle fifo {
usecase "put" as put
usecase "get" as get
usecase "reset" as reset
usecase "is_empty" as is_empty
usecase "is_full" as is_full
}
producer --> put
consumer --> get
producer --> reset
consumer --> is_full
consumer --> is_empty
@enduml
```

Listing 1: usecase.plantuml



Use Case Diagram of a First-in-First-out Queue

The test cases were created from the template given in the assignment. The implemented test cases can be seen in Appendix A
The code was uploaded to github and can be found here: https://github.com/Steinarr134//EmbeddedAssignments/tree/master/Assignment3_2TestDrivenDevelopment

Part 2

The given templates were copied into the correct positions. `fifo.h` was renamed to `fifo2.h` so it and `fifo2.cpp` could be distinguished from `fifo3.h` and `fifo3.cpp`. That way the solutions to Part 2 and Part 3 could be programmed without interfering with each other.

A First-In-First-Out Queue was implemented by using a vector called `buffer` and when an item is 'gotten' the `Fifo` class pops the first item and then moves all the items one step forward. The class keeps count of how many items are in the queue with a private attribute called `count` and therefore knows where to put items that get 'put' to it.

```
#include "fifo2.h"

Fifo::Fifo()
{
    reset();
}

int Fifo::get()
{
    // get shouldn't be called on an empty queue but:
    if (is_empty())
    {
        return -1;
    }

    int ret = buffer[0]; // get the return item

    // move evrything down by one seat
    for (int i = 1; i<count; i++)
    {
        buffer[i-1] = buffer[i];
    }
    count--; // there will be one fewer items in the queue now
    return ret;
}

void Fifo::put(int item)
{
    if (is_full())
    {
        get(); // throw one out to make room
        put(item); //recursive for the win!
        return;
    }
    // put the item in the queue and then increment count
    buffer[count++] = item;
}

bool Fifo::is_empty()
{
    return count == 0;
}

bool Fifo::is_full()
{
    return count==FIFO_SIZE;
}

void Fifo::reset()
{
    count = 0;
}
```

Listing 2: src/fifo2.cpp - Implementation of a Fifo class using a vector buffer.

A main program was written that accepts characters from the Serial port and puts them into the queue. Every second one character is printed from the queue and the onboard LED is turned on while the queue is full. The code for `main.cpp` can be seen in Appendix B

Part 3

A second implementation of the `Fifo` class was written based on a ring buffer. The class uses private attributes `head` and `tail` that are pointers to where in the ring buffer the next item to be 'gotten' is and where the next item to be 'put' should go, respectively.

```
#include "fifo3.h"

Fifo::Fifo()
{
    reset();
}

int* Fifo::incr_p(int* p){
    // this function increments the pointers in the ring buffer
    if (p + 1 == &buffer[0] + FIFO_SIZE)

    // if we're about to exit the buffer
        return &buffer[0]; // return to 0
    else
        return p+1;
}

int Fifo::get()
{
    // get shouldn't be called on an empty queue but:
    if (is_empty())
    {
        return -1;
    }

    int ret = *head; // get first item
    head = incr_p(head); // move the head by 1

    // if the queue is now empty:
    if (head == tail) it_is_empty = true;

    return ret;
}

void Fifo::put(int item)
{
    if (is_full())
    {
        get();  // throw one out to make room
        put(item); //recursive for the win!
        return;
    }
    *tail = item; // put uten into queue
    tail = incr_p(tail); // increment tail
    it_is_empty = false; // no longer empty
}
```

```
bool Fifo::is_empty()
{
    return it_is_empty;
}

bool Fifo::is_full()
{
    return (head == tail && !it_is_empty);
}

void Fifo::reset()
{
    tail = &buffer[0];
    head = &buffer[0];
    it_is_empty = true;
}
```

Listing 3: src/fifo3.cpp - Implementation of a Fifo class using a ring buffer and pointers.

Appendix

A Test cases

```
#include <avr/delay.h>
#include <unity.h>

#include <fifo3.h>

void test_normal_flow(void)
{
    // 1 Setup
    Fifo f;
    f.put(1);
    f.put(2);
    f.get();
    f.put(3);

    // 2-3 Execute and validate
    TEST_ASSERT_EQUAL(2, f.get());
    TEST_ASSERT_EQUAL(3, f.get());

    f.put(1);
    f.put(2);
    f.put(3);
    f.put(4);
    f.put(5);
    f.get();

    // 4 Cleanup
}

void test_underflow(void)

{
    // 1 Setup
    Fifo f;
    f.put(1);
    f.put(2);
    f.get();
    f.get();

    // 2-3 Execute and validate
    TEST_ASSERT_TRUE(f.is_empty());

    // 4 Cleanup
}

void test_reset(void)
{
    Fifo f;
    f.put(1);
    f.put(3);
    TEST_ASSERT_FALSE(f.is_empty());
    f.reset();
    TEST_ASSERT_TRUE(f.is_empty());
}

void test_overwrite()
```

```
{
    Fifo f;
    for (int i = 0; i<FIFO_SIZE; i++)
    {
        f.put(i+1);
    }
    TEST_ASSERT_TRUE(f.is_full());
    // buffer = [1, 2, 3, 4, 5]
    // now the queue is full so putting one more should result in the oldest one
    (1) popping out
    f.put(100);
//     // thus the oldest member is now 2
    // f.get();
    TEST_ASSERT_EQUAL(2, f.get());
}

void test_overflow()
{
    Fifo f;
    TEST_ASSERT_FALSE(f.is_full());
    for (int i = 0; i<FIFO_SIZE; i++)
    {
        f.put(i);
    }
    TEST_ASSERT_TRUE(f.is_full());
}

void basic()
{

    TEST_ASSERT_TRUE(false);
}

int main()
{
    // NOTE!!! Wait for >2 secs
    // if board doesn't support software reset via Serial.DTR/RTS
    _delay_ms(2000);

    UNITY_BEGIN(); // IMPORTANT LINE!count==FIFO_SIZE;

    RUN_TEST(test_normal_flow);
    RUN_TEST(test_underflow);
    RUN_TEST(test_reset);
    RUN_TEST(test_overflow);
    RUN_TEST(test_overwrite);
    // RUN_TEST(basic);

    UNITY_END(); // stop unit testing
}
```

Listing 4: test/test_main.cpp

B main.cpp

```
#include <Arduino.h>
#include "fifo3.h" // change to fifo2.h for part 2
#include "digital_out.h"
// State Behaviour based on the C++ example at Refactoring Guru
// nothing happens in loop, we never get there.
void loop() {}

void setup()
{
  Serial.begin(115200); // open serial port
  Fifo f;
  Digital_out led(5);
  delay(100);
  Serial.println("Assignment 3_2 - FIFO");
  char incoming;
  long last_time = 0;

  while (true)
  {

    if (Serial.available() > 0)
    {
      // read the incoming byte:
      incoming = Serial.read();

      // put into queue
      f.put(incoming);
      // if queue is full -> turn on led
      if (f.is_full()){
        led.set_hi();
      }
    }
    // print one item from queue every second
    if (millis() - last_time > 1000){
      last_time = millis();
      if (!f.is_empty())
      {
        Serial.println((char)f.get());
      }
      led.set_lo(); // set led low, queue isn't full anymore
    }

  }

}
```

Listing 5: src_main.cpp

C Header files

```

#ifndef FIFO_H
#define FIFO_H

const int FIFO_SIZE = 5;

class Fifo
{
public:
    Fifo();
    int get();
    void put(int item);
    bool is_empty();
    bool is_full();
    void reset();

private:
    int buffer[FIFO_SIZE];
    int count;
};

#endif // FIFO_H
```

Listing 6: include/fifo2.h

```

#ifndef FIFO_H
#define FIFO_H

const int FIFO_SIZE = 5;

class Fifo
{
public:
    Fifo();
    int get();
    void put(int item);
    bool is_empty();
    bool is_full();
    void reset();

private:
    int buffer[FIFO_SIZE];
    int* head;
    int* tail;
    int* incr_p(int* p); // for incrementing the pointers
    bool it_is_empty;
};

#endif // FIFO_H
```

Listing 7: include/fifo3.h