2. Blinky.cpp - D3

```
#include "mraa.h"
#include <stdio.h>
int main()
{
        mraa_gpio_context d_pin = NULL;
        d_pin = mraa_gpio_init(13);

        if (d_pin == NULL) {
                fprintf(stderr, "MRAA couldn't initialize GPIO, exiting");
                return MRAA_ERROR_UNSPECIFIED;
        }
        if (mraa_gpio_dir(d_pin, MRAA_GPIO_OUT) != MRAA_SUCCESS) {
                fprintf(stderr, "Can't set digital pin as output, exiting");
                return MRAA_ERROR_UNSPECIFIED;
        };
        for (int i=10;i>0;i--) {
                printf("LED OFF\n");
                mraa_gpio_write(d_pin, 0);
                sleep(1);
                printf("LED ON\n");
                mraa_gpio_write(d_pin, 1);
                sleep(1);
        }
        return MRAA_SUCCESS;
}
```

3. For only Button - Button.cpp - D4

```
#include "grove.h"
#include <stdio.h>
#include <unistd.h>
int main()
{
  upm::GroveButton* button = new upm::GroveButton(4);
  int count = 5;
  int button_val=0;
  while( count > 0 ) {
        button_val = button->value();
        printf ("Program will exit after %d button presses\n", count);
        printf ("Button value is: %d\n ", button_val);
     if (button_val)
        count--;
     usleep(500000);
  }
        printf ("Exiting, bye!");
  delete button;
}
```

For both Button and Light (Button - D2 and Light - D3) Blinky.Cpp

```
#include “mraa.h”
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#define BUTTON_PIN 2
#define LED_PIN 13
int main()
{
mraa.init();
mraa.gpio.context button=mraa.gpio init(BUTTON_PIN);
mraa_gpio_dir(button,MRAA_GPIO_IN);
mraa_gpio_context led = mraa_gpio_init(LED_PIN);
mraa_gpio_dir(led, MRAA_GPIO_OUT);
for (;;) {
        if (mraa_gpio_read((button)) mraa_gpio_write(led, 1);
        else mraa_gpio_write(led, 0);
        unsleep(100000);
}
mraa_gpio_close(button);
mraa_gpio_close(led);
mraa_deinit();
return 0;
}
```

4. Buzzer.cpp - D5

```
#include <buzzer.hpp>
#include <stdio.h>
#include <unistd.h>
int main()
{
   int chord[] = { DO, RE, MI, FA, SOL, LA, SI, DO, SI };
          upm::Buzzer* sound = new upm::Buzzer(5);
          printf("Volume = %f\n", sound->getVolume());
          sound->setVolume(0.6);
          printf("Volume = %f\n", sound->getVolume());
          fflush(stdout);
          printf("\nPlaying notes, pausing for 0.1 seconds between notes...\n");
          fflush(stdout);
          for (int chord_ind = 0; chord_ind < 7; chord_ind++) {
             printf(" %d\n",  sound->playSound(chord[chord_ind], 500000) );
             usleep(100000);
          }
          printf("Exiting, bbye!\n");
          delete sound;
}
```

5. Temperature.cpp - A0

```
#include "mraa/aio.h"
#include <math.h>
#include <stdio.h>
#include <unistd.h>
#include "jhd1313m1.h"
#include "grove.h"
int main()
{
    mraa_aio_context adc_a0;
    uint16_t adc_value = 0;
    const int B=4275;
    const int R0 = 100000;
    adc_a0 = mraa_aio_init(0);
    if (adc_a0 == NULL) {
        return 1;
    }
    for (int i=10; i>0;i--) {
        adc_value = mraa_aio_read(adc_a0); //Max value @ 5V = 1024
        printf("ADC A0 read value : %d\n", adc_value);
        float R = 1023.0/((float)adc_value)-1.0;
        R = 100000.0*R;
        float temperature=1.0/(log(R/100000.0)/B+1/298.15)-273.15;
        printf("Temperature value : %.2f Degree Celsius\n", temperature);
        sleep(1);
    }
    mraa_aio_close(adc_a0);
    printf("Exiting .. Bbye!");
    return MRAA_SUCCESS;
}
```

6. TouchInterrupt.cpp - D4

```
#include "ttp223.h"
#include <stdio.h>
#include <unistd.h>
void touchISR (void*);
int count = 5;
void touchISR(void*)
{
        count--;
        printf("\nHello World from ISR, will exit after %d touch events", count);
        fflush(stdout);
}
int main()
{
  upm::TTP223* touch = new upm::TTP223(4);
  touch->installISR(mraa::EDGE_FALLING, &touchISR, NULL);
        printf("\nWelcome, waiting for touch event.\nWill exit after 5 events");
        fflush(stdout);
  while(count>0);
  printf("\nExiting .. Bbye!");
  delete touch;
}
```

7. Light.cpp - A0

```
#include "grove.h"
#include <stdio.h>
#include <unistd.h>
int main()
{
	upm::GroveLight* light = new upm::GroveLight(0);
	for (int i=20;i>0;i--) {
		printf(" Light value is %f which is roughly %d lux \n", light->raw_value(), light->value());
		fflush(stdout);
		sleep(1);
	}
	printf("Exiting .. bbye!");
	delete light;
}
```

8. Mic.cpp - A0

```
#include "mic.h"
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
int is_running = 1;
uint16_t buffer [128];
upm::Microphone *mic = NULL;
void sig_handler(int signo)
{
    printf("got signal\n");
    if (signo == SIGINT) {
        is_running = 0;
    }
}
int main(int argc, char **argv)
{
    mic = new upm::Microphone(0);
    if (signal(SIGINT, sig_handler) == SIG_ERR)
            printf("\ncan't catch SIGINT\n");
    thresholdContext ctx;
    ctx.averageReading = 0;
    ctx.runningAverage = 0;
    ctx.averagedOver   = 2;
    while (is_running) {
        int len = mic->getSampledWindow (2, 128, buffer);
        if (len) {
            int thresh = mic->findThreshold (&ctx, 30, buffer, len);
            mic->printGraph(&ctx);
            if (thresh) {
                // do something ....
            }
        }
    }
    printf ("exiting application\n");
    delete mic;
    return 0;
}
```

9. LCD.cpp - last I2C which is close to D5

```
#include "jhd1313m1.h"
#include <stdio.h>
#include <unistd.h>
int main(void)
{
        upm::Jhd1313m1 *lcd;
  lcd = new upm::Jhd1313m1(0, 0x3E, 0x62);
  printf("Display text on LCD\n");
  lcd->setCursor(0,0);

  lcd->write("Batch 4");

  lcd->setCursor(1,2);

  lcd->write("welcome !");

  printf("Sleeping for 5 seconds\n");
  sleep(5);
  printf("Starting Color loop...\n");

  for (int i = 5; i>0 ;i--){
  lcd->setColor(255,220,220);
  sleep(1);
  lcd->setColor(0,255,0);
  sleep(1);
  lcd->setColor(0,0,125);
  sleep(1);
  }
  printf("Exiting .. bbye!\n");
  delete lcd;
  return 0;
}
```

10 . RotaryAngle.cpp - A1

```
#include "grove.h"
#include <stdio.h>

#include <unistd.h>
int main()
{
        upm::GroveRotary* knob = new upm::GroveRotary(1);
  while( 1 ) {
      float abs_value = knob->abs_value(); // Absolute raw value
      float abs_deg = knob->abs_deg();     // Absolute degrees
      float abs_rad = knob->abs_rad();     // Absolute radians
      float rel_value = knob->rel_value(); // Relative raw value
      float rel_deg = knob->rel_deg();     // Relative degrees
      float rel_rad = knob->rel_rad();     // Relative radians
      printf("Absolute: %4d raw %5.2f deg = %3.2f rad Relative: %4d raw %5.2f deg %3.2f rad\n",
          (int16_t)abs_value, abs_deg, abs_rad, (int16_t)rel_value, rel_deg, rel_rad);
      sleep(1);
    }
  delete knob;
}
```