```
#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(0.1);
    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;
}
```

```c
#include "mraa/aio.h" //for mraa_aio_read()
#include <math.h> // for math functions
#include <stdio.h> // for printf()
#include <unistd.h> //for sleep()

int main()
{
    mraa_aio_context adc_a0;
    uint16_t adc_value = 0;

    const int B=4275;                 // B value of the thermistor
    const int R0 = 100000;            // R0 = 100k

    adc_a0 = mraa_aio_init(0);

    if (adc_a0 == NULL) {
        return 1;
    }
    for (int i=5; 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;
//convert to temperature as per datasheet

        printf("Temperature value : %.2f Degree Celsius\n",
temperature);

        sleep(1);
    }
    mraa_aio_close(adc_a0);
    printf("Exiting .. Bbye!");
    return MRAA_SUCCESS;
}
```

```
#include "ttp223.h" // for button->value()
#include <stdio.h> // for printf()
#include <unistd.h> //for sleep()

void touchISR (void*);
int count = 3;

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;
    return MRAA_SUCCESS;
}
```

```
#include "grove.h" //for light->name(), light->value()
#include <stdio.h> // for printf()
#include <unistd.h> //for sleep()

int main()
{
        upm::GroveLight* light = new upm::GroveLight(0);
        for (int i=5;i>0;i--) {
            printf(" Light value is %f which is roughly %d lux \n",
light->raw_value(), light->value());
            fflush(stdout);
            sleep(1);
        }
        // Delete the light sensor object
        printf("Exiting .. bbye!");
        delete light;
}
```

```
#include "mic.h" // for
#include <stdio.h> // for printf()
#include <unistd.h> //for sleep()
#include <signal.h>
#include <sys/time.h>

int is_running = 1;
uint16_t buffer [128];    //define buffer to store captures values

upm::Microphone *mic = NULL;    //create microphone object

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;

}
```

```
#include "jhd1313m1.h"
/*for
    lcd->setCursor();
    lcd->write();
    lcd->setCursor();
    lcd->setColor();
 */
#include <stdio.h> // for printf()
#include <unistd.h> //for sleep()

int main(void)
{
    // 0x62 RGB_ADDRESS, 0x3E LCD_ADDRESS
    upm::Jhd1313m1 *lcd;
    lcd = new upm::Jhd1313m1(0, 0x3E, 0x62);
    //arguments: I2C addresses of LCD controller and LED backlight
controller

    printf("Display text on LCD\n");

    lcd->setCursor(0,0);    //bring cursor to top left corner
    lcd->write("23: Rohini");    //print text
    lcd->setCursor(1,0);    //bring cursor to second row
    lcd->write("35: Devansh");    //print text

    printf("Sleeping for 10 seconds\n");
    sleep(10);
    printf("Starting Color loop...\n");

    //Run loop for toggling backlight color between Red->Green->Blue
x 5 times
    for (int i = 5; i>0 ;i--){
    lcd->setColor(255,0,0);
    sleep(1);
    lcd->setColor(0,255,0);
    sleep(1);
    lcd->setColor(0,0,255);
    sleep(1);
    }

    printf("Exiting .. bbye!\n");

    delete lcd;    //free up memory.
    return 0;
}
```

```
#include "grove.h" // for button->value()
#include <stdio.h> // for printf()
#include <unistd.h> //for sleep()

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;
    return MRAA_SUCCESS;
}
```