```
#include "Wire.h"
#include <QueueList.h>
```

```
#define P0  2
#define P1  3
#define P2  4
#define P3  5
#define P4  6
#define P5  7
#define P6  8
#define P7  9
#define P8  10
#define P9  11
#define P10  12
#define P11  13
#define P12  A0
#define P13  A1
#define P14  A2
#define P15  A3
```

Define I/O pins

```
enum Protocol {
  HEADER,
  COMMAND,
  RESET,
  STATUS,
  SIZE,
  END
};
```

สร้าง enum ใช้เป็น state ของ protocol

```
uint8_t dataSize;
uint8_t command;
uint8_t inputByte;
uint8_t high;
uint8_t low;
unsigned int index;
```

ประกาศตัวแปรที่ใช้ทั้งหมด

```
QueueList <uint8_t> queue;
Protocol protocolUart;
```

สร้าง queue เพื่อเก็บ protocol
สร้าง protocol state

```
class Locker {
  public:
    uint8_t pinNum;
    // 0 = close 1 = open
    bool Status;

    Locker() {}

    void check() {
      Status = digitalRead(pinNum);
    }
};

Locker locker[16];
```

สร้าง class locker เพื่อไว้เก็บค่า pin I/O และ สถานะเปิดปิด

```
  Serial.begin(9600);
  pinMode(P0, INPUT_PULLUP);
  pinMode(P1, INPUT_PULLUP);
  pinMode(P2, INPUT_PULLUP);
  pinMode(P3, INPUT_PULLUP);
  pinMode(P4, INPUT_PULLUP);
  pinMode(P5, INPUT_PULLUP);
  pinMode(P6, INPUT_PULLUP);
  pinMode(P7, INPUT_PULLUP);
  pinMode(P8, INPUT_PULLUP);
  pinMode(P9, INPUT_PULLUP);
  pinMode(P10, INPUT_PULLUP);
  pinMode(P11, INPUT_PULLUP);
  pinMode(P12, INPUT_PULLUP);
  pinMode(P13, INPUT_PULLUP);
  pinMode(P14, INPUT_PULLUP);
  pinMode(P15, INPUT_PULLUP);
```

กำหนด pin I/O ทุกช่องเป็น input pull up

```
  protocolUart = HEADER;
  for ( int i = 0; i < 16; i++) {
    locker[i].pinNum = i + 2;
    locker[i].check();
    Serial.print("pin num is: ");
    Serial.println(locker[i].pinNum);
    Serial.print("status is: ");
    Serial.println(locker[i].Status);
  }
  index = 0;
```

กำหนด state แรก

เช็คค่าของทุกช่อง

```
  Wire.begin(0x71);
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
```

onReceive ไว้รับค่าจาก master
onRequest ไว้ส่งค่าให้ master

```
void receiveEvent(int howMany) {

  while (1 < Wire.available()) {
    uint8_t c = Wire.read(); // 
    queue.push(c);
    // print the character
  }
  uint8_t x = Wire.read();
  queue.push(x);

void getProtocol() {
  if (queue.isEmpty()) {
    return;
  }

  switch (protocolUart) {
    case HEADER :
      inputByte = queue.pop();
      Serial.println("HEADER");
      if (inputByte == 0x02) {
        protocolUart = COMMAND;
      }
      break;

    case COMMAND :
      inputByte = queue.pop();
      Serial.println("COMMAND");
      Serial.println(inputByte);
      if (inputByte == 0x21) {
        command = 0x21;
      }
      else if (inputByte == 0x22) {
        command = 0x22;
      }
      else {
        protocolUart = HEADER;
      }
      protocolUart = SIZE;
      break;
```

```
  case SIZE :
    Serial.println("SIZE");
    dataSize = queue.pop();
    Serial.print("size = ");
    Serial.println(dataSize);

    if (command == 0x21) {
      if (dataSize == 0x01) {
        protocolUart = RESET;
      }
      else {
        protocolUart = HEADER;
      }
    }
    else if (command == 0x22) {
      if (dataSize == 0x01) {
        protocolUart = STATUS;
      }
      else {
        protocolUart = HEADER;
      }
    }
    break;

  case RESET :
    inputByte = queue.pop();
    if (inputByte == 0x00) {
      Serial.println("RESET");
      protocolUart = END;
    } else {
      protocolUart = HEADER;
    }
    break;
   case STATUS :
     inputByte = queue.pop();
     if (inputByte == 0x00) {
       Serial.println("STATUS");
       protocolUart = END;
     } else {
       protocolUart = HEADER;
     }
     break;

   case END :
     inputByte = queue.pop();
     if (inputByte == 0x03) {
       Serial.println("END");
       processProtocol();
     }
     protocolUart = HEADER;
     break;
}
```

```
void processProtocol() {
  if (command == 0x21) {
    Serial.println("process reset");
    resetFunc();
  }
  else if (command == 0x22) {
    Serial.println("process status");
    readStatus();
    checkStatus();
  }
}

void readStatus() {
  for (int i = 0; i < 16; i++) {
    locker[i].check();
  }
}

void checkStatus() {
  index = 0;
  Serial.println("checking status");
  for (int i = 15; i >= 0; i--){
    index = (index << 1);
    index += locker[i].Status;
    Serial.print("index = ");
    Serial.println(index, BIN);
  }

  Serial.print("index = ");
  Serial.println(index, BIN);
  low = index & 0xff;
  Serial.print("low = ");
  Serial.println(low, BIN);
  high = (index >> 8);
  Serial.print("high = ");
  Serial.println(high, BIN);
}

void requestEvent() {
  Wire.write(high);
  Serial.println(high);
  Wire.write(low);
  Serial.println(low);
}
```

เช็คสถานะของทุกช่อง

ทุกครั้งที่เก็บค่าจากช่องนึง ให้ shift bit ไปทางขวา 1 bit ก่อนเก็บค่าต่อไป

แบ่งค่าจาก 16 bits 1 byte เป็น 8 bits 2 bytes
low เอามา & กับ “1111 1111”
ส่วน high เอา 8 bits ที่เหมือมาใส่ โดย shift bit มาด้านซ้ายมา 8 bits

ส่งค่าทั้ง 2 bytes ให้ master