ДОДАТОК А

Лістинг програми

```
#ifndef TIMES_H
#define TIMES_H

static const uint8_t countMealTimes = 5;



struct Times {
  uint8_t hours;
  uint8_t minutes;
  uint8_t seconds;

  Times() {
    hours = -1;
    minutes = -1;
    seconds = -1;
  }

  Times(uint8_t hours, uint8_t minutes, uint8_t seconds) {
    this->hours = hours;
    this->minutes = minutes;
    this->seconds = seconds;
  }

  bool isTimeSet() {
    return hours != -1;
  }

  bool isTimeEqual(uint8_t hours, uint8_t minutes, uint8_t seconds) {
    return this->hours == hours && this->minutes == minutes && this->seconds ==
seconds;
  }
};

struct MealTimes {
  Times meals[5];
};
#endif // TIMES_H



#include <DS1307.h>
#include <EEPROM.h>
#include "times.h"

const uint8_t addressIsTimeSet = 0;

class Settings {
  public:
    bool isTimeSetup() {
      return this->_isTimeSet;
    }
```

```
bool isMealTimesSetup() {
  return this->_isMealTimesSet;
}
bool isTimerOpenFlapSetup() {
  return this->_isTimerOpenFlapSet;
}
bool isSetup() {
  return _isTimeSet && _isMealTimesSet && _isTimerOpenFlapSet;
}
void setupTime(uint8_t hours, uint8_t minutes, uint8_t seconds) {
  _rtc->setTime(hours, minutes, seconds);

  this->_isTimeSet = true;
  EEPROM.put(addressIsTimeSet, this->_isTimeSet);
}
void setupMealTime(Times* meals) {
  uint8_t addressIsMealTimesSet = _getAddressForIsMealTimesSet();

  MealTimes mealTimes;

  for(uint8_t i = 0; i < countMealTimes; i++) {
    mealTimes.meals[i] = *(meals + i);
  }

  _isMealTimesSet = true;
  _mealTimes = mealTimes;

  EEPROM.put(addressIsMealTimesSet, _isMealTimesSet);
  EEPROM.put(_getAddressForMealTimes(), mealTimes);
}

void setupTimerOpenFlap(int timeInMillis) {
  _isTimerOpenFlapSet = true;
  _timerOpenFlap = timeInMillis;

  EEPROM.put(_getAddressForIsTimerOpenFlapSet(), _isTimerOpenFlapSet);
  EEPROM.put(_getAddressForTimerOpenFlap(), _timerOpenFlap);
}

MealTimes getMealTimes() {
  return this->_mealTimes;
}

Times* getMeal(uint8_t index) {
  return &_mealTimes.meals[index];
}

int getTimerOpenFlap() {
  return this->_timerOpenFlap;
}

void clearEEPROM() {
  for (int i = 0 ; i < EEPROM.length() ; i++) {
    EEPROM.write(i, 0);
  }
}
```

```
    MealTimes _getMealTimesFromEEPROM() {
      MealTimes mts;
      uint8_t address = _getAddressForMealTimes();

      EEPROM.get(address, mts);

      return mts;
    }

    Settings(DS1307 *rtc) {

      _rtc = rtc;
      _isTimeSet = _getIsTimeSetFromEEPROM();
      _isMealTimesSet = _getIsMealTimesSetFromEEPROM();
      _isTimerOpenFlapSet = _getIsTimerOpenFlapSetFromEEPROM();
      _timerOpenFlap = _getTimerOpenFlapFromEEPROM();
      _mealTimes = _getMealTimesFromEEPROM();
    }

  private:
    bool _isTimeSet = false;
    bool _isMealTimesSet = false;
    bool _isTimerOpenFlapSet = false;
    int _timerOpenFlap = 0;
    MealTimes _mealTimes;
    DS1307 *_rtc;

    uint8_t _getAddressForIsMealTimesSet() {
      return sizeof(bool); //its "isTimeSet"
    }

    uint8_t _getAddressForIsTimerOpenFlapSet() {
      return _getAddressForIsMealTimesSet() + sizeof(bool); // its isMealTimeSet
    }

    uint8_t _getAddressForTimerOpenFlap() {
      return _getAddressForIsTimerOpenFlapSet() + sizeof(bool); // its
IsTimerOpenFlapSet;
    }


    uint8_t _getAddressForMealTimes() {
      return _getAddressForTimerOpenFlap() + sizeof(int); // its TimerOpenFlap
    }

    bool _getIsTimeSetFromEEPROM() {
      bool isTimeSet = false;

      EEPROM.get(addressIsTimeSet, isTimeSet);

      return isTimeSet;
    }

    bool _getIsMealTimesSetFromEEPROM() {
      bool isMealTimesSet = false;

      EEPROM.get(_getAddressForIsMealTimesSet(), isMealTimesSet);
```

```
      return isMealTimesSet;
    }

    bool _getIsTimerOpenFlapSetFromEEPROM() {
      bool isTimerOpenFlapSet = false;

      EEPROM.get(_getAddressForIsTimerOpenFlapSet(), isTimerOpenFlapSet);

      return isTimerOpenFlapSet;
    }

    uint8_t _getTimerOpenFlapFromEEPROM() {
      uint8_t timerOpenFlap;

      EEPROM.get(_getAddressForTimerOpenFlap(), timerOpenFlap);

      return timerOpenFlap;
    }
};


class Button {
  public:
    Button (byte pin) {
      _pin = pin;
      pinMode(_pin, INPUT_PULLUP);
    }
    bool isClicked() {
      bool btnState = digitalRead(_pin);
      if (!btnState && !_flag && millis() - _timer >= 10) {
        _flag = true;
        _timer = millis();
        return true;
      }
      if (!btnState && _flag && millis() - _timer >= 30) {
        _timer = millis();
        return true;
      }
      if (btnState && _flag) {
        _flag = false;
        _timer = millis();
      }
      return false;
    }
  private:
    byte _pin;
    uint32_t _timer;
    bool _flag;
};
```

```
#include <OLED_I2C.h>
#include "button.h"
#include "settings.h"
#include "feeder.h"

class Menu {
  public:
    void setupTime() {
      if (!_settings->isTimeSetup()) {
        _oled->clrScr();

        if (!isTempTime) {
          _oled->print("Greeting", CENTER, 5);
          _oled->print("set current time", CENTER, 20);
          _oled->print("press 'OK' for save", CENTER, 35);
          _printWaitingCursorForTime();
          _changePartTimeSetup();
          _setupPartTime();
        }
        else {
          _oled->print("Is time correct?", CENTER, 10);
          _oled->print(_getStrTime(), CENTER, 25);
          _printWaitingCursorForYesNo();
          _changePartYesNoSetup();
          _setupTime();
        }
        if (buttonOk.isClicked()) {
          isTempTime = true;
        }
      }
    }

    void setupMealTimes() {
      if (!_settings->isMealTimesSetup() && _settings->isTimeSetup()) {
        _oled->clrScr();
        if (!isTempTime && currentMealTimeSetup <= 5) {
          _oled->print("Set meal time, max 5", CENTER, 5);
          _oled->print("current time: " + String(currentMealTimeSetup), CENTER,
20);
          _oled->print("press 'OK' for save", CENTER, 35);
          _printWaitingCursorForTime();
          _changePartTimeSetup();
          _setupPartTime();
        }
        else {
          _oled->print("Is time correct?", CENTER, 10);
          _oled->print(_getStrTime(), CENTER, 25);
          _printWaitingCursorForYesNo();
          _changePartYesNoSetup();
          _addMealTime();
          _setupMealTimes();
        }
        if (buttonOk.isClicked()) {
          isTempTime = true;
        }
      }
```

```
    }

    void setupTimerOpenFlap() {
      if (!_settings->isTimerOpenFlapSetup() && _settings->isMealTimesSetup()) {
        _oled->print("set time open flap", CENTER, 15);
        _oled->print("press 'OK' for save", CENTER, 35);
        _printWaitingCursorForTimerTest();
        _changePartTimerOrTest();
        _setupPartTime(&isTimer, &timerOpenFlap, 1000);
        _setupTimerOpenFlap();
      }
    }

    void clearEEPROM() {
      _settings->clearEEPROM();
    }

    void currentTime(uint8_t hours, uint8_t minutes, uint8_t seconds) {
      _oled->print(_getStrWithTime(hours) + " : " + _getStrWithTime(minutes) + "
: " + _getStrWithTime(seconds), CENTER, 30);
    }

    Menu(OLED *oled, Settings *settings, Feeder *feeder) {
      this->_oled = oled;
      this->_settings = settings;
      this->_feeder = feeder;
    }
  private:
    OLED* _oled;
    Settings* _settings;
    Feeder* _feeder;

    Button buttonRight = Button(6);
    Button buttonLeft = Button(4);
    Button buttonUp = Button(5);
    Button buttonDown = Button(3);
    Button buttonOk = Button(2);

    bool isSetHours = 1;
    bool isSetMinutes = 0;
    bool isSetSeconds = 0;

    bool isTempTime = 0;
    bool isYes = 1;
    bool isNo = 1;
    bool isTimer = 1;
    bool isTest = 0;

    uint8_t currentMealTimeSetup = 1;
    int timerOpenFlap = 0;
    uint8_t hours = 0;
    uint8_t minutes = 0;
    uint8_t seconds = 0;
    Times mealTimes[5];

    void _setupTime() {
      if (isYes && buttonOk.isClicked()) {
```

```
        _settings->setupTime(hours, minutes, seconds);
        _oled->clrScr();
        isTempTime = false;
      }
      if (isNo && buttonOk.isClicked()) {
        isTempTime = false;
      }
    }

    void _addMealTime() {
      if (isYes && buttonOk.isClicked()) {
        Times newMealTime(hours, minutes, seconds);
        mealTimes[currentMealTimeSetup - 1] = newMealTime;

        currentMealTimeSetup++;
        isTempTime = false;
      }
      if (isNo && buttonOk.isClicked()) {
        isTempTime = false;
      }

    }

    void _setupMealTimes() {
      if (currentMealTimeSetup > 5) {
        _settings->setupMealTime(mealTimes);
        currentMealTimeSetup = 1;
        _oled->clrScr();
      }
    }

    void _setupTimerOpenFlap() {
      if (isTimer && buttonOk.isClicked()) {
        _settings->setupTimerOpenFlap(timerOpenFlap);
        _oled->clrScr();
      }
      if (isTest) {
        isTest = !_feeder->openFlap(timerOpenFlap);
        isTimer = !isTest;
      }
    }

    bool _isPassedMillis(int mills) {
      return round(millis() / mills) % 2 == 0;
    }

    void _printWaitingCursorForTime() {
      _printWaitingCursor(_getStrForTimeWithCursor(_getPositionTime()),
_getStrTime());
    }

    void _printWaitingCursorForYesNo() {
      _printWaitingCursor(_getStrYesNoWithCursor(_getPositionYesNo()),
_getStrYesNo());
    }

    void _printWaitingCursorForTimerTest() {
```

```
_printWaitingCursor(_getStrForTimerOpenFlapWithCursor(_getPositionTimerTest()),
_getStrForTimerOpenFlap());
    }

    void _printWaitingCursor(String strWithCursor, String strWithoutCursor) {
      if (_isPassedMillis(100)) {
        _oled->print(strWithCursor, CENTER, 50);
      }
      else {
        _oled->print(strWithoutCursor, CENTER, 50);
      }
    }

    String _getStrForTimeWithCursor(uint8_t position) {
      if (position < 0 || position > 2)
        "position must be >= 0 and <= 2";

      return _getOpenCharForCursor(position, 0) + _getStrWithTime(hours) +
_getCloseCharForCursor(position, 0) + ":"
      + _getOpenCharForCursor(position, 1) + _getStrWithTime(minutes) +
_getCloseCharForCursor(position, 1) + ":"
      + _getOpenCharForCursor(position, 2) + _getStrWithTime(seconds) +
_getCloseCharForCursor(position, 2);
    }

    String _getStrTime() {
      return " " + _getStrWithTime(hours) + " : " + _getStrWithTime(minutes) + "
: " + _getStrWithTime(seconds) + " ";
    }

    String _getStrForTimerOpenFlapWithCursor(uint8_t position) {
      return _getOpenCharForCursor(position, 0) + String(timerOpenFlap) +
_getCloseCharForCursor(position, 0) + "ms   "
       + _getOpenCharForCursor(position, 1) + "test" +
_getCloseCharForCursor(position, 1);
    }

    String _getStrYesNoWithCursor(uint8_t position) {
      return _getOpenCharForCursor(position, 0) + String("Yes") +
_getCloseCharForCursor(position, 0) + "/"
      + _getOpenCharForCursor(position, 1) + String("No") +
_getCloseCharForCursor(position, 1);
    }

    String _getStrYesNo() {
      return " Yes / No ";
    }

    String _getStrForTimerOpenFlap() {
      return " " + String(timerOpenFlap) + " ms    test ";
    }

    char _getOpenCharForCursor(uint8_t position, uint8_t expectedPosition) {
      return position == expectedPosition ? '<' : ' ';
    }
```

```
char _getCloseCharForCursor(uint8_t position, uint8_t expectedPosition) {
  return position == expectedPosition ? '>' : ' ';
}

uint8_t _getPositionTime() {
  if (isSetHours)
    return 0;
  if (isSetMinutes)
    return 1;
  return 2;
}

uint8_t _getPositionYesNo() {
  if (isYes)
    return 0;
  return 1;
}

uint8_t _getPositionTimerTest() {
  if (isTimer)
    return 0;
  return 1;
}

String _getStrWithTime(uint8_t time) {
  String strTime = String(time);

  if (strTime.length() == 2)
    return strTime;
  return "0" + strTime;
}

void _setupPartTime() {
  _setupPartTime(&isSetHours, &hours, 23);
  _setupPartTime(&isSetMinutes, &minutes, 59);
  _setupPartTime(&isSetSeconds, &seconds, 59);
}

void _setupPartTime(bool *flag, uint8_t *countTime, uint8_t max) {
  if (*flag) {
    if (buttonUp.isClicked()) {
      if (*countTime < max)
        *countTime = *countTime + 1;
      else
        *countTime = 0;
    }
    if (buttonDown.isClicked()) {
      if (*countTime > 0)
        *countTime = *countTime - 1;
      else
        *countTime = max;
    }
  }
}

void _setupPartTime(bool *flag, int *countTime, int max) {
  if (*flag) {
```

```
        if (buttonUp.isClicked()) {
          if (*countTime < max)
            *countTime = *countTime + 5;
          else
            *countTime = 0;
        }
        if (buttonDown.isClicked()) {
          if (*countTime > 0)
            *countTime = *countTime - 5;
          else
            *countTime = max;
        }
      }
    }

    void _changePartTimeSetup() {
      if (isSetHours)
        _changeLeftRightMenuPart(&isSetHours, &isSetSeconds, &isSetMinutes);
      if (isSetMinutes)
        _changeLeftRightMenuPart(&isSetMinutes, &isSetHours, &isSetSeconds);
      if (isSetSeconds)
        _changeLeftRightMenuPart(&isSetSeconds, &isSetMinutes, &isSetHours);
    }

    void _changePartYesNoSetup() {
      if (isYes)
        _changeLeftRightMenuPart(&isYes, &isNo, &isNo);
      if (isNo)
        _changeLeftRightMenuPart(&isNo, &isYes, &isYes);
    }

    void _changePartTimerOrTest() {
      if (isTimer)
        _changeLeftRightMenuPart(&isTimer, &isTest, &isTest);
      if (isTest)
        _changeLeftRightMenuPart(&isTest, &isTimer, &isTimer);
    }

    void _changeLeftRightMenuPart(bool *currentPart, bool *leftPart, bool
*rightPart) {
      if (buttonRight.isClicked()) {
        *currentPart = false;
        *rightPart = true;
      }
      if (buttonLeft.isClicked()) {
        *currentPart = false;
        *leftPart = true;
      }
    }
};
```

```
#include <Servo.h>
#include "menu.h"
#include "fountain.h"

extern uint8_t SmallFont[];

uint8_t pinServo = 7;

Servo servo;
DS1307 rtc(A4, A5);
OLED oled(A2, A3);
Feeder feeder(&rtc, &servo);
Fountain fountain(1, 2, 3, &oled);
Settings settings(&rtc);
Menu menu(&oled, &settings, &feeder);

void setup()
{
  Serial.begin(9600);
  Serial.println("start setup");
  setupOled();
  setupServo();
  setupMealTimes();
}

void setupOled() {
  oled.begin();
  oled.setFont(SmallFont);
  oled.clrScr();
  oled.update();
  Serial.println("oled setup successful");
}

void setupServo() {
  servo.attach(pinServo);
  servo.write(closeDegree);
  Serial.println("servo setup successful");
}

void loop()
{
  if (settings.isSetup()) {
    workflow();
  }
  else {
    menu.setupTime();
    menu.setupMealTimes();
    menu.setupTimerOpenFlap();
    setupMealTimes();
  }

  oled.update();
}

void workflow() {
  currentTime();
  feeder.workflow();
```

```
  fountain.workflow();
}

void currentTime() {
  Time now = rtc.getTime();
  menu.currentTime(now.hour, now.min, now.sec);
}

void setupMealTimes() {
  feeder.updateMealTimes(settings.getMealTimes());
  feeder.updateTimerOpenFlap(500);
}
```

ДОДАТОК Б

Демонстраційний матеріал

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кіль к.	Примітки
			Текстові документи		
А4	1	ГЮІК.482389.002 ПЗ	Пояснювальна записка	1	75 ст. ф. А;
			Інші документи		
	2		Презентаційний матеріал	1	17 ст. ф. *.ppt
			Лістинг програмного коду	1	12 ст. ф. А4

					ГЮІК.482389.002 ВД			
Змін.	Арк.	Номер докум.	Підпис	Дата				
Розроб.		Водянов А.А.			Система автоматизованої годівниці для домашніх тварин	Літера	Аркуш	Аркушів
Перевір.		Цимбал О.М.				Н	1	1
Н.контр.		Демська Н.П.			Відомість кваліфікаційної роботи	Кафедра КІТАР ХНУРЕ		
Затв.		Невлюдов І.Ш.						