ROBOCODE
SENIOR EMBEDDED

ООП. ВЛАСНА БІБЛІОТЕКА

LESSON 28. FOR STUDENTS

English Time

Час Англійської

Library	Бібліотека
Constructor	Конструктор
Arguments	Аргументи
Overload	Перевантаження

Рівні доступу

i Існують наступні основні рівні доступу до даних в класі

i **public** - дані, що можна отримати та змінювати поза класом

i **private** - дані, що можна отримати і змінити тільки в середині класу

i **protected** - дані, що можна змінювати та читати тільки в середині класу, або дочірньому класу

? Визначіть рівень доступу даних класу “собака”: лапи, гавкати, колір

Рівні доступу

i

Як буде виглядати такі рівні доступу в вигляді коду

```
class Dog {
  public:
    int paws = 4;
    void gaw();
  private:
    String color;
};

void Dog::gaw() {
  // тут описуємо як гавкає собака
}
```

Конструктор класу

Перед використанням класу, ми створювали об’єкт цього класу. Проте можна створювати об’єкт класу із визначеними параметрами

Ми можемо встановити необмежену кількість параметрів, перед створенням об’єкту класу

Для цього необхідно створити **конструктор** класу

```
Dog(String color, String name){

}
```

Конструктор має ту ж назву, що і клас, а також він не має визначеного типу даних

Конструктор класу

i Ми створювали до цього клас “собака”, а потім визначали певні його параметри. А тепер представте, що буде, якщо ви забудете визначити колір, або ім’я вашому об’єкту.

i Використання конструктору дає можливість створювати об’єкти з обов’язковими параметрами

i Тепер перед створюванням об’єкту, ми зможемо вказати обов’ язкові параметри

```
Dog Rex("White","Rex");
```

Конструктор класу

Нижче наведено створений конструктор класу та створення об’єкту із зазначеними параметрами

```
class Dog {
  public:
    Dog(String color, String name) {
      col=color;
      nam=name;
    }
    String col,nam;
    int paws = 4;
    void gaw();
  private:
    String color;
};

Dog Rex("White","Rex");
```

Конструктор класу Sonar

Створіть самостійно в класі Sonar конструктор, що буде отримувати параметри портів, перед запуском. Визначіть об’єкт класу згідно нового конструктора

```
class Sonar {
  public:
    Sonar(int echo, int trig) {
      echoPin = echo;
      trigPin = trig;
    }
    void begin();
    int distance();
    void readDistance();
  private:
    int echoPin;
    int trigPin;
    long dur, dist;
    int maxRange = 300;
    int minRange = 0;
};

Sonar sr(13, 12);
```

Перевантаження

Існує така потреба, щоб один метод або конструктор міг викликатися із різними параметрами та їх типами даних та міг виконувати різні дії

Ми знаємо про те, що не можна створювати два однакові методи з однаковими назвами. Проте, якщо цей метод матиме різний тип даних параметрів, або різну кількість параметрів, то при викликанні методу, спрацює той, що відповідає цим параметрам

```
void sleep(int time) {
   Serial.print(time);
   Serial.print("hrrrrr");
 }
 void sleep(float time) {
   Serial.print(time);
   Serial.print("hrrrrr");
 }
 void sleep(int time, String hr){
   Serial.print(time);
   Serial.print(hr);
 }
```

Перевантаження

i

Перевантажувати можна методи, оператори та конструктори

i

Для класу Sonar створіть перевантаження конструктору класу, де другий конструктор буде додатково визначати швидкість монітору порту, яку можна було б налаштувати за потреби

Перевантаження

```
class Sonar {
  public:
    Sonar(int echo, int trig) {
      echoPin = echo;
      trigPin = trig;
    }
    Sonar(int echo, int trig, int bod) {
      echoPin = echo;
      trigPin = trig;
      spd = bod;
    }
    void begin();
    int distance();
    void readDistance();
  private:
    int echoPin;
    int trigPin;
    int spd;
    long dur, dist;
    int maxRange = 300;
    int minRange = 0;
};

void Sonar::begin() {
  Serial.begin(spd);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

Sonar sr(13, 12, 115200);
```

Бібліотека

i Великі проекти незручно тримати в одному файлі за проблеми великої кількості коду

i Тому, створюють бібліотеки, кожна з якої може вирішити певну задачу

i Кожна бібліотека складається із двох файлів - той, що виконує **(.cpp)** та файл заголовків **(.h)** . Така структура потрібна для більш швидкого пошуку потрібної функції та її виконання

Файл .cpp

i

В цьому файлі описується сам клас та його реалізація. Файл, що виконується. Тут необхідно підключити всі файли заголовків та інших бібліотек, що можуть використовуватися та вказати всі методи, що використовуються

```
#if defined(ARDUINO_ARCH_AVR)
#include <avr/interrupt.h>
#include <Arduino.h>

#include "Servo.h"

#define usToTicks(_us)    (( clockCyclesPerMicrosecond()* _us)
/ 8)     // converts microseconds to tick (assumes prescale of
8)  // 12 Aug 2009
#define ticksToUs(_ticks) (( (unsigned)_ticks * 8)/
clockCyclesPerMicrosecond() ) // converts from ticks back to
microseconds


#define TRIM_DURATION       2                                   //
compensation ticks to trim adjust for digitalWrite delays // 12
August 2009

//#define NBR_TIMERS        (MAX_SERVOS / SERVOS_PER_TIMER)

static servo_t servos[MAX_SERVOS];                              //
static array of
```

Файл .h

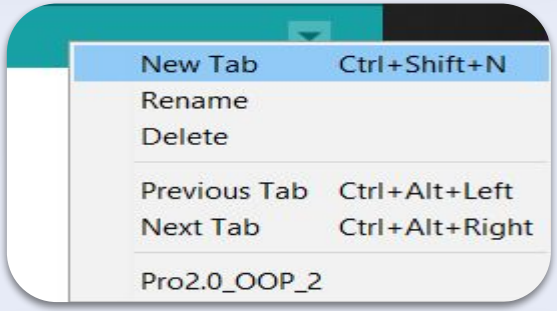
Файл заголовків схожий на лист змісту в книзі. Тут записані заголовки методів та їх параметри, а також всі змінні, але їх реалізація відсутня. Саме цей файл підключається в якості файлу бібліотеки

```
#ifndef Servo_h
#define Servo_h

#include <inttypes.h>

class Servo
{
public:
  Servo();
  uint8_t attach(int pin);           // attach the given pin to
the next free channel, sets pinMode, returns channel number or
0 if failure
  uint8_t attach(int pin, int min, int max); // as above but
also sets min and max values for writes.
  void detach();
  void write(int value);             // if value is < 200 its
treated as an angle, otherwise as pulse width in microseconds
  void writeMicroseconds(int value); // Write pulse width in
microseconds
  int read();                        // returns current pulse
width as an angle
};
#endif
```

Створюємо власну бібліотеку

Тепер спробуємо перенести клас Sonar в файл **.cpp** та створити власну бібліотеку!

Для цього створимо два файли. Перший *Sonar.cpp* та *Sonar.h*



В файлі Sonar.cpp підключимо бібліотеку **Arduino.h** та **Sonar.h**

```
#include <Arduino.h>
#include "Sonar.h"
```

Файл Sonar.cpp

i

В даному файлі необхідно описати тільки методи та конструктори, що були раніше створені

i

Ми звикли не підключати Arduino.h, в кожній програмі, оскільки ArduinoIDE робить це самостійно за нас, але коли ми створюємо окрему бібліотеку або просто файл, необхідно підключати цю бібліотеку, для доступу до всіх стандартних команд Arduino

Файл Sonar.cpp

```cpp
#include "Sonar.h"

Sonar::Sonar(int echo, int trig) {
  echoPin = echo;
  trigPin = trig;
}
Sonar::Sonar(int echo, int trig, int bod) {
  echoPin = echo;
  trigPin = trig;
  spd = bod;
}

void Sonar::begin() {
  Serial.begin(spd);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

int Sonar::distance() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  dur = pulseIn(echoPin, HIGH);
  dist = dur / 58.2;
  if (dist >= maxRange || dist <= minRange)
  {
    return 0;
  }
  else
  {
    return dist;
  }
}

void Sonar::readDistance() {
  Serial.println(distance());
}
```

Файл Sonar.h

В даний файл помістимо все, що стосується файлу, проте конструктор має містити тільки заголовок

Для уникнення дублювання кількох бібліотек із однаковими назвами створюється препроцесорна перевірка за допомогою препроцесорних команд: #ifndef, #define, #endif

Для захисту назви найчастіше використовується назва бібліотеки великими буквами, нижнє підкреслення, формат файлу

```
#ifndef SONAR_H
#define SONAR_H
//YOUR CODE
#endif
```

Файл Sonar.h

i

Код без препроцесорної перевірки і захисту імені

```
#include <Arduino.h>

class Sonar {
  public:
    Sonar(int echo, int trig);
    Sonar(int echo, int trig, int bod);
    void begin();
    int distance();
    void readDistance();
  private:
    int echoPin;
    int trigPin;
    int spd;
    long dur, dist;
    int maxRange = 300;
    int minRange = 0;
};
```

Файл Sonar.h

i

Код з препроцесорною перевіркою і захисту імені

```
#ifndef SONAR_H
#define SONAR_H
#include <Arduino.h>
class Sonar {
  public:
    Sonar(int echo, int trig);
    Sonar(int echo, int trig, int bod);
    void begin();
    int distance();
    void readDistance();
  private:
    int echoPin;
    int trigPin;
    int spd;
    long dur, dist;
    int maxRange = 300;
    int minRange = 0;
};

#endif
```

Файл Sonar.ino

Тепер перевіримо нашу бібліотеку, створивши мінімальну для цього програму в виконуваному файлі

```
#include "Sonar.h"

Sonar sr(13, 12, 115200);

void setup() {
  sr.begin();
}

void loop() {
  sr.readDistance();
}
```

Файл keywords.txt

Ми помічали, як методи та різні змінні підсвічуються різними кольорами. В ArduinoIDE це реалізується в файлі keywords.txt, де можна визначити підсвічування потрібних назв та слів

KEYWORD1 (DataType)	`Sonar`
KEYWORD2 (Methods and Functions)	`begin()`
LITERAL1 (Constants)	`pin`

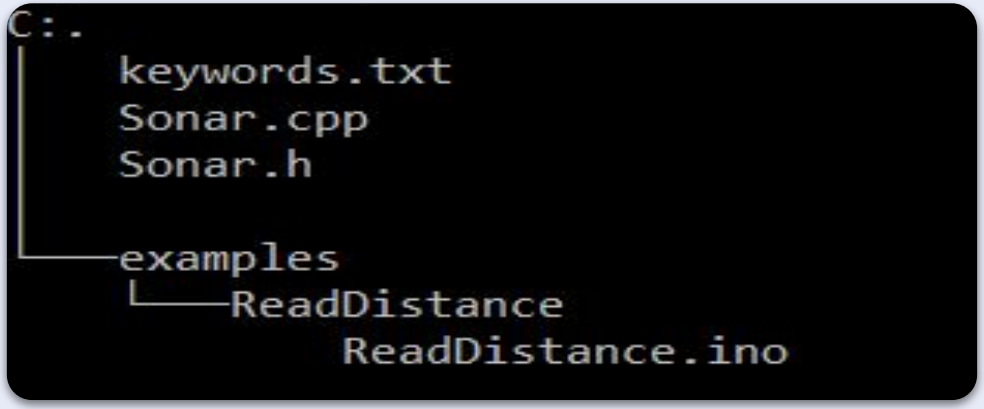
Структура бібліотеки

i

Кожна бібліотека має схожу структуру. Ми перейменуємо наш .ino файл в **ReadDistance.ino**

i

Створимо папку SonarLib та наступну структуру нашого проекту



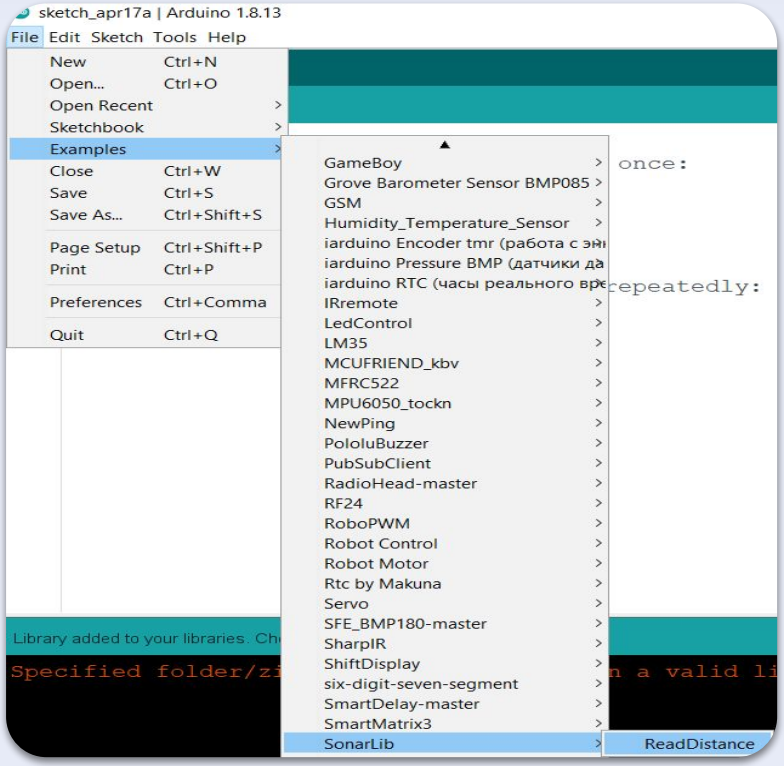
Структура бібліотеки

- Помістимо папку з бібліотекою до інших бібліотек
- Загальний шлях ./Documents/Arduino/libraries/
- Щоб було зручно передавати власну бібліотеку, її можна архівувати у **.zip** архів

Структура бібліотеки

i

Перезапустимо Arduino IDE і знайдемо приклад з нашої бібліотеки



Структура бібліотеки

Вітаю! Ви створили власну бібліотеку!