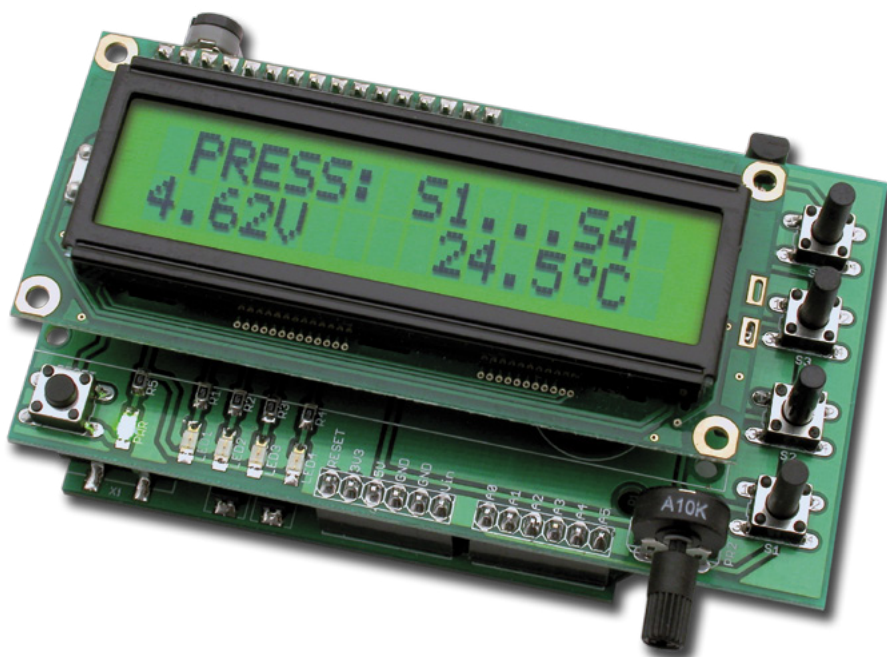


# Kurs programowania Arduino (6)



## Obsługa wyświetlacza LCD, cyfrowych linii I/O i przetwornika A/C

W tej części kursu zostaną opisane praktyczne przykłady użycia oprogramowania Arduino. Zaznajomienie się z nimi ułatwi tworzenie programu a dodatkowo, gotowe przykłady można po niewielkich modyfikacjach zastosować we własnych projektach. Przykłady zostały przygotowane dla zestawu Arduino UNO oraz współpracujących z nim modułów AVTDuino LCD i AVTDuino LED.



### Obsługa interfejsu RS232

Arduino UNO nie ma wyświetlacza. Najprościej do komunikacji z użytkownikiem użyć interfejsu komunikacyjnego RS232 i programu terminala, w który wyposażono Arduino IDE. Komunikacja za pomocą interfejsu szeregowego przebiega z zastosowaniem konwertera USB-RS232 i linii TXD/RXD mikrokontrolera. Do komunikacji za pośrednictwem sprzętowego interfejsu RS232 są dostępne następujące funkcje:

- `begin()` – ustalenie prędkość transmisji,
- `end()` – wyłączenie transmisji szeregowy,
- `available()` – pobranie liczby otrzymanych bajtów,
- `read()` – odbiór ramki danych,
- `peek()` – odczyt kolejnego znaku z bufora,
- `flush()` – opróżnienie bufora odbiornika,
- `print()` – wysłanie znaku,
- `println()` – wysyłanie ciągu znaków z kodami końca i nowej linii,
- `write()` – wysłanie bajtu.

Zazwyczaj często używa się funkcji `print()` lub `println()` umożliwiających przesłanie ciągu znaków. Kilka przykładów zamieszczono niżej:

```
Serial.print(78, BYTE) da "N"
Serial.print(78, BIN) da
"1001110"
Serial.print(78, OCT) da "116"
```

```
Serial.print(78, DEC) da "78"
Serial.print(78, HEX) da "4E"
Serial.println(1.23456, 0) da "1"
Serial.println(1.23456, 2) da
"1.23"
```

Opcjonalny drugi parametr tej funkcji wskazuje na typ danych. Na **listingu 1** pokazano przykładowy program obsługujący komunikację za pomocą interfejsu szeregowego. Program wysyła tablice znaków ASCII. Ma również możliwość odebrania kodu znaku którego następnie odsyła w formie dziesiętnej i szesnastkowej.

W pierwszej kolejności w programie za pomocą funkcji `Serial.begin(9600)` konfigurowana jest prędkość transmisji na 9600 bodów. Funkcja `Serial.println` drukuje nagłówek tablicy ASCII. Następnie zmienna `bajt` o początkowej wartości 33 wskazuje początek kodów ASCII. Następnie w programie wykonywane są instrukcje w pętli `while(bajt<127)`, która jest wykonywana aż wartość bajt jest mniejsza od 127. W pętli wykonywane są instrukcje drukujące kody tablicy ASCII w formacie tekstowej, dziesiętnej oraz

**Dodatkowe materiały na CD/FTP:**  
<ftp://ep.com.pl>, user: 15352, pass: 760hp6s5  
 • poprzednie części kursu

szesnastkowej. Po każdym wykonaniu instrukcji wysyłających dane wykonywana jest instrukcja `bajt++` zwiększająca o 1 wartość zmiennej `bajt`. Po wysłaniu tablicy wykonywane są instrukcje w nieskończonej pętli `while(1)`. W warunku `if (Serial.available() > 0)` za pomocą funkcji `available()` sprawdzane jest czy został otrzymany jakikolwiek znak. Wtedy funkcja zwraca wartość większą od 0. Jeśli tak jest wykonywana jest instrukcja `bajt = Serial.read()` odbierająca otrzymaną wartość i zapisuje ją do zmiennej `bajt`. W kolejnych instrukcjach wysyłana jest otrzymana wartość w `bajt` w formie dziesiętnej oraz szesnastkowej. Na **rysunku 1** pokazano działanie programu obsługującego komunikację szeregową z użyciem interfejsu RS232.

### Obsługa wyświetlacza LCD

Każdy system musi wskazywać użytkownikowi swoje działanie i rezultaty obliczeń.

**Listing 1. Przykładowy program obsługujący komunikację za pomocą interfejsu szeregowego**

```

void setup() //procedura konfiguracyjna
{
  Serial.begin(9600); //konfigurowanie prędkości transmisji
  Serial.println("Tablica kodów ASCII"); //wysłanie tekstu
}

int bajt = 33; //zmienna kodów ASCII z wartością początkową tablicy 33
void loop() //procedura programu głównego
{
  while(bajt<127) //pętla wykonywana aż bajt mniejszy od 127
  {
    Serial.print(bajt, BYTE); //wysłanie znaku z wartością BYTE
    Serial.print(" ", dec: " "); //wysłanie opisu wartości DEC
    Serial.print(bajt); //wysłanie znaku
    Serial.print(" ", hex: "0x"); //wysłanie opisu HEX
    Serial.println(bajt, HEX); //wysłanie wartości w formie HEX
    bajt++; //zwiększenie o 1 wartości zmiennej bajt
  }
  while(1) //początek nieskończonej pętli
  {
    if (Serial.available() > 0) { //jeśli odebrano bajt danych z interfejsu
      RS232 to
      bajt = Serial.read(); //odczyt odebranego znaku do zmiennej bajt
      Serial.print("dec: "); //wysłanie opisu
      Serial.print(bajt); //wysłanie odebranego znaku
      Serial.print(" ", hex: "0x"); //wysłanie opisu HEX
      Serial.println(bajt, HEX); //wysłanie odebranego znaku w formie HEX
    }
  }
}

```

**Listing 2. Przykładowy program obsługujący wyświetlacz LCD**

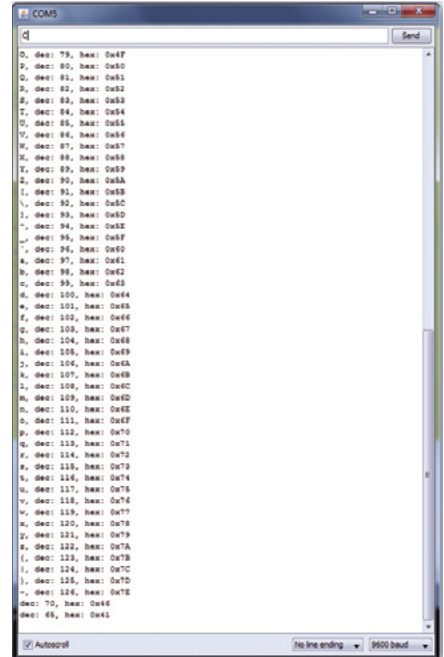
```

#include <LiquidCrystal.h> //biblioteka obsługi LCD
LiquidCrystal lcd(8, 9, 4, 5, 6, 7); //konfigurowanie linii do których
dołączono LCD
int wart_temp=24; //przykładowa zmienna z zapisaną wartością temperatury
byte st[8] = { //tablica znaku stopnia dla wyświetlacza LCD
  B00100,
  B01010,
  B00100,
  B00000,
  B00000,
  B00000,
  B00000,
  B00000,
};

void setup() { //funkcja inicjalizacji
  lcd.begin(16, 2); //konfiguracja rozdzielczości wyświetlacza LCD
  lcd.createChar(0, st); //funkcja utworzenia własnego znaku z tablicy st
  o kodzie 0
}

void loop() { //pętla główna programu
  lcd.clear(); //czyszczenie LCD
  lcd.setCursor(2, 0); //ustawienie kursora w 2 kolumnie pierwszego wiersza
  lcd.print("Arduino w EP"); //wyświetlenie napisu
  lcd.setCursor(0, 1); // 1 kolumna, 2 wiersz
  lcd.print("Temp: "); //wyświetlenie na LCD napisu Temp:
  lcd.print(wart_temp); //wyświetlenie zmiennej wart_temp
  lcd.write(0); //wyświetlenie znaku stopnia
  lcd.print("C"); //wyświetlenie znaku C
  lcd.setCursor(12, 1); //12 kolumna, 2wiersz
  lcd.cursor(); //włączenie kursora
  lcd.blink(); //włączenie migającego kursora
  while(1); //pętla nieskończona
} //koniec pętli głównej

```

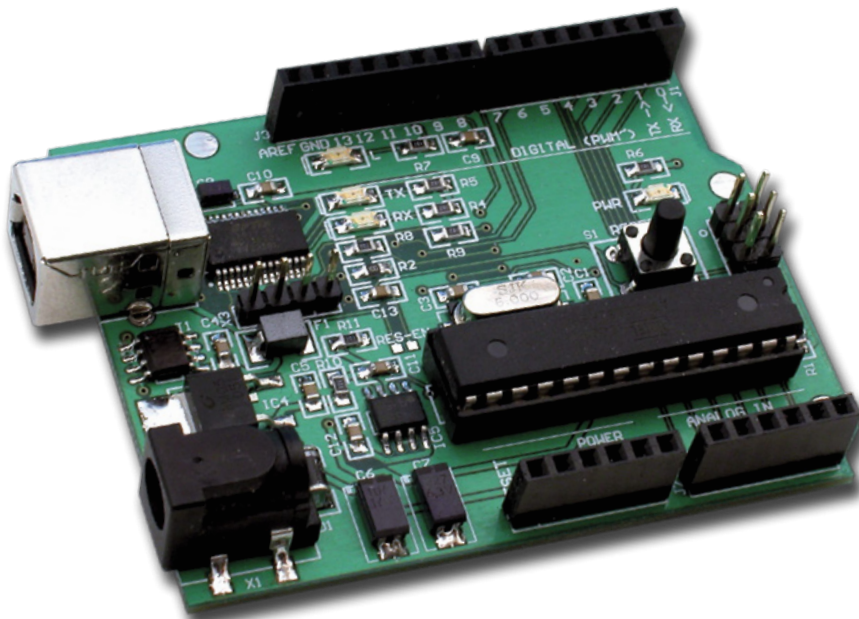


**Rysunek 1. Ekran programu terminala z danymi przesłanymi za pomocą RS232**

Bardzo często do tego celu są wykorzystywane wyświetlacze. Dlatego też zostanie pokazana obsługa wyświetlacza LCD w Arduino z wykorzystaniem modułu AVTduino LCD. Moduł ten wyposażono w wyświetlacz, kilka przycisków oraz diody LED.

Do obsługi wyświetlaczy z kontrolerem HD44780 w Arduino IDE jest przeznaczona biblioteka *LiquidCrystal (LCD)* która działa ze sterowaniem LCD 4-bitowym lub 8-bitowym. Ma ona następujące funkcje:

- *LiquidCrystal()* – definiuje piny do których został dołączony LCD,
- *begin()* – definiuje rozdzielczość zastosowanego LCD,
- *clear()* – czyści ekran LCD,
- *home()* – ustawia kursor na początku ekranu LCD,
- *setCursor()* – ustawia kursor w zadanym miejscu LCD,
- *write()* – zapisuje znak do LCD,
- *print()* – zapisuje znak lub znaki do LCD,
- *cursor()* – włącza kursor,
- *noCursor()* – wyłącza kursor,
- *blink()* – włącza migający kursor,
- *noBlink()* – wyłącza migający kursor,
- *display()* – włącza ekran LCD,
- *noDisplay()* – wyłącza ekran LCD,
- *scrollDisplayLeft()* – przesuwa zawartość LCD w lewo,
- *scrollDisplayRight()* – przesuwa zawartość LCD w prawo,
- *autoscroll()* – automatyczne przesuwanie zawartości na LCD,
- *noAutoscroll()* – wyłączenie automatycznego przesuwania zawartości na LCD,
- *leftToRight()* – ustawia kierunek zapisu tekstu od prawej do lewej,
- *rightToLeft()* – ustawia kierunek zapisu tekstu od lewej do prawej,



- `createChar()` – umożliwia definicję własnego znaku.

Przykładowy program obsługujący wyświetlacz LCD pokazany został na **listingu 2**. Wyświetla on na wyświetlaczu kilka przykładowych komunikatów wraz z przykładem wyświetlenia własnego zdefiniowanego znaku.

W pierwszej kolejności w programie wykonywana jest instrukcja `#include <LiquidCrystal.h>` dołączająca do programu bibliotekę obsługującą LCD. Za pomocą funkcji `LiquidCrystal lcd(8, 9, 4, 5, 6, 7)` są konfigurowane linie, do których dołączono wyświetlacz. Zmienna `wart_temp` o wartości domyślnej „24” zawiera liczbę, która będzie interpretowana jako temperatura 24°C. W tablicy `byte st[8]` zdefiniowano znak stopnia. Tablica ta odwzorowała wygląd znaku, gdzie bit ustawiony oznacza piksel włączony, a wyzerowany – wyłączony. W funkcji konfiguracyjnej jest wywoływana procedura `lcd.begin(16, 2)` określająca typ wyświetlacza. Wyświetlacz został skonfigurowany jako 2 wiersze po 16 znaków. Funkcja `lcd.createChar(0, st)` przesyła do wyświetlacza definicję znaku zapisanego w tablicy `st` o kodzie 0 (znak stopnia).

Umieszczona na początku programu głównego instrukcja `lcd.clear()` czyści ekran LCD. Następnie, funkcja `lcd.setCursor(2, 0)` ustawia jest kursor w pierwszej linii i trzeciej kolumnie. Dalej, za pomocą funkcji `lcd.print()` jest wyświetlany jest w pierwszej linii LCD przykładowy tekst. Następnie kursor zostaje ustawiony w drugiej linii LCD i jest wyświetlany komunikat `Temp:` oraz wartość zmiennej `wart_temp`. Funkcja `lcd.write(0)` wyświetla wcześniej zdefiniowany znak stopnia o kodzie 0. Na końcu programu zostaje wyświetlony kursor, a za pomocą komendy `lcd.blink()` jest włączane jego migotanie.

## Obsługa cyfrowych linii I/O

Obsługa linii cyfrowych mikrokontrolera w głównej mierze polega na konfiguracji ich czy mają pracować jako wyjścia lub wejścia z rezystorem podciągającym. Do konfiguracji kierunku pracy linii cyfrowej służy komenda `pinMode()`, do zapisu stanu linii `digitalWrite()` a odczytu `digitalRead()`. Na **listingu 3** pokazano przykład programu który obsługuje wyjścia do których dołączono diody LED, generator PIEZO oraz 4 przyciski które są dostępne w module AVTDuino LCD. Przycisk S1 powoduje zapalenie LED1, przycisk S2 miganie LED2. Po naciśnięciu przycisku S3 generowany jest kod Morse’A SOS a naciśnięcie S4 generuje efekt świetlny biegnącego światła.

Za pomocą komendy `const` przypisano nazwy wyprowadzeniom I/O, do których dołączono diody, przyciski oraz brzęczyk. Linie, do których dołączono diody oraz brzęczyk są skonfigurowane jako wyjściowe, a linie do

### Listing 3. Przykładowy program do obsługi linii I/O

```
const int Led1 = 13; //definiowanie aliasów
const int Led2 = 12;
const int Led3 = 11;
const int Led4 = 10;
const int SW1 = 3;
const int SW2 = 2;
const int SW3 = 1;
const int SW4 = 0;
const int Buzzer = A5;

void setup() { //funkcja inicjalizacji
  pinMode(Led1, OUTPUT); //Konfiguracja linii do których są dołączone diody
  jako wyjścia
  pinMode(Led2, OUTPUT);
  pinMode(Led3, OUTPUT);
  pinMode(Led4, OUTPUT);
  pinMode(Buzzer, OUTPUT); //linia I/O sterująca brzęczyk
  pinMode(SW1, INPUT); //linie wejściowe przycisków
  pinMode(SW2, INPUT);
  pinMode(SW3, INPUT);
  pinMode(SW4, INPUT);
  digitalWrite(SW1, HIGH); //dołączenie rezystorów podciągających
  digitalWrite(SW2, HIGH);
  digitalWrite(SW3, HIGH);
  digitalWrite(SW4, HIGH);
  digitalWrite(Led1, HIGH); // wyłączenie diód LED
  digitalWrite(Led2, HIGH);
  digitalWrite(Led3, HIGH);
  digitalWrite(Led4, HIGH);
  digitalWrite(Buzzer, HIGH); // wyłączenie brzęczyka
}

void loop() { //pętla główna
  if (digitalRead(SW1) == LOW) { //Sprawdzenie czy naciśnięty S1
    digitalWrite(Led1, LOW); //zaświecenie LED1
    while(digitalRead(SW1) == LOW); //oczekiwanie na zwolnienie S1
  }
  else { //w przeciwnym razie
    digitalWrite(Led1, HIGH); //wyłączenie diody LED1
  }

  if (digitalRead(SW2) == LOW) { //Sprawdzenie czy naciśnięty S2
    while(digitalRead(SW2) == LOW) //jeśli wciśnięty
    {
      digitalWrite(Led2, LOW); //włączenie LED2
      delay(200); //opóźnienie 200 ms
      digitalWrite(Led2, HIGH); //wyłączenie LED2
    }
  }
}
```

REKLAMA



## Złączki SMD seria 2060

### Drzwi do świata LED

- wersje 1-, 2- i 3-biegunowe
- wysokość 4,5 mm pozwala zredukować cienie w aplikacjach LED
- do montażu powierzchniowego SMT na płytkach drukowanych
- łatwa obsługa dzięki wbudowanemu przyciskowi
- do wszystkich typów przewodów 0,2 – 0,75 mm<sup>2</sup>
- opakowanie typu „tape and reel”
- grupowanie złączek w wielobiegunowe listwy, bez utraty rastra

50-506 Wrocław, ul. Piękna 58a, [www.wago.com](http://www.wago.com)

**WAGO**<sup>®</sup>  
INNOVATIVE CONNECTIONS

których dołączono przyciski – wejściowe. Dodatkowo, do linii wejściowych za pomocą komend `digitalWrite()` dołączono rezystory podciągające. Przyciśnięcie przycisku spowoduje wyzerowanie poziomu na wejściu.

W pętli głównej programu, instrukcja warunkowa `if (digitalRead(SW1) == LOW)` sprawdza czy naciśnięto przycisk S1. Jeśli tak, to przez wyzerowanie odpowiedniej linii zostaje zaświecona dioda LED1. W pętli `while(digitalRead(SW1) == LOW)` program czeka na zwolnienie przycisku S1. Po jego zwolnieniu, dioda LED1 zostaje zgaszona (instrukcje w klauzuli `else`). W kolejnym warunku jest rozpatrywany stan przycisku S2. Po jego wciśnięciu dioda LED2 migocze, za co odpowiada pętla `while(digitalRead(SW2) == LOW)`. Po wciśnięciu przycisku S3 jest generowany sygnał świetlny oraz dźwiękowy SOS. Instrukcje w pierwszej pętli `for` generują trzy kropki, w kolejnej trzy kreski, a w ostatniej ponownie trzy kropki. Generowanie sygnału SOS powtarza się co 2 sekundy (instrukcja `delay(2000)`). Po przyciśnięciu przycisku S4, za pomocą diod LED1...LED4, jest wyświetlany pewien efekt świetlny.

## Obsługa linii analogowych

Mikrokontroler zestawu Arduino UNO ma wejścia analogowe oznaczone A0... A5. Wartości napięć doprowadzone do nich mogą być zmierzone przez przetwornik A/C. Na listingu 4 zamieszczono program mierzący i wyświetlający napięcie z potencjometru znajdującego się w module AVTduino LCD i dołączonego do linii A0.

W pierwszej kolejności tworzone są zmienne, w których będzie zapamiętana wartość zmierzona i odczytana z przetwornika A/C oraz napięcie obliczone na jej podstawie. Z wykorzystaniem funkcji `analogReference(DEFAULT)` konfigurowane jest napięcie odniesienia dla przetwornika A/C. Wybrano wartość domyślną, to jest 5 V – napięcie zasilające. Tym samym zakres mierzonych napięć zostaje ustalony na 0...5 V. Komenda `analogRead(A0)` uruchamia pomiar napięcia na linii A0 i odczyt zmierzonej wartości z przetwornika A/C. Wartość ta jest zapamiętywana w zmiennej `wart_pot`. Na jej podstawie wylicza się wartość zmierzonego napięcia za pomocą wyrażenia `wart_nap=(5.0*wart_pot)/1024.0`, w którym wartość „5” to napięcie odniesienia dla A/C, natomiast „1024” to rozdzielczość przetwornika. Obliczona wartość napięcia jest zapamiętywana w zmiennej `wart_nap`. W kolejnych instrukcjach, wartości odczytana i obliczona są wyświetlane na drugiej linii wyświetlacza LCD. Każda zmiana położenia osi potencjometru będzie powodować zmianę napięcia mierzonego przez przetwornik A/C.

**Marcin Wiązania, EP**

### Listing 3. c.d.

```

    delay(200); //opóźnienie 200 ms
  }
}
else { //w przeciwnym razie
  digitalWrite(Led2, HIGH); //wyłączenie diody LED2
}
if (digitalRead(SW3) == LOW) { //sprawdzenie czy naciśnięty S3
  while(digitalRead(SW3) == LOW) //jeśli wciśnięty
  {
    for (int i=0; i<3; i++) { //pętla generująca trzy kropki
      digitalWrite(Led3, LOW); //włączenie LED3
      digitalWrite(Buzzer, LOW); //włączenie brzęczyka
      delay(150); // opóźnienie 150 ms
      digitalWrite(Led3, HIGH); // wyłączenie LED3
      digitalWrite(Buzzer, HIGH); //wyłączenie brzęczyka
      delay(100); // opóźnienie 100 ms
    }
    for (int i=0; i<3; i++) { //pętla generująca trzy kreski
      digitalWrite(Led3, LOW); //włączenie LED3
      digitalWrite(Buzzer, LOW); //włączenie brzęczyka
      delay(400); //opóźnienie 400 ms
      digitalWrite(Led3, HIGH); //wyłączenie LED3
      digitalWrite(Buzzer, HIGH); //wyłączenie brzęczyka
      delay(100); // opóźnienie 100 ms
    }
    for (int i=0; i<3; i++) { //pętla generująca trzy kropki
      digitalWrite(Led3, LOW); //włączenie LED3
      digitalWrite(Buzzer, LOW); //włączenie brzęczyka
      delay(150); // opóźnienie 150 ms
      digitalWrite(Led3, HIGH); //wyłączenie LED3
      digitalWrite(Buzzer, HIGH); //wyłączenie brzęczyka
      delay(100); //opóźnienie 100 ms
    }
    delay(2000); //opóźnienie 2 sekundy
  }
}
else { //w przeciwnym razie
  digitalWrite(Led3, HIGH); //wyłączenie diody LED3
  digitalWrite(Buzzer, HIGH); //wyłączenie brzęczyka
}
if (digitalRead(SW4) == LOW) { //sprawdzenie czy wciśnięty S4
  while(digitalRead(SW4) == LOW) //jeśli przyciśnięty
  {
    digitalWrite(Led1, LOW); //włączenie LED1
    delay(100); //opóźnienie 100 ms
    digitalWrite(Led2, LOW); //włączenie Led2
    delay(100);
    digitalWrite(Led3, LOW); //włączenie LED3
    delay(100);
    digitalWrite(Led4, LOW); //włączenie LED4
    delay(100);
    digitalWrite(Led1, HIGH); //wyłączenie LED1
    delay(100);
    digitalWrite(Led2, HIGH); //wyłączenie Led2
    delay(100);
    digitalWrite(Led3, HIGH); //wyłączenie LED3
    delay(100);
    digitalWrite(Led4, HIGH); //wyłączenie LED4
    delay(100);
  }
}
delay(300);
} //koniec pętli głównej programu

```

### Listing 4. Przykładowy program mierzący napięcie na wyjściu potencjometru

```

#include <LiquidCrystal.h> //biblioteka obsługi LCD
LiquidCrystal lcd(8, 9, 4, 5, 6, 7); //konfigurowanie LCD
int wart_pot; //zmienna na wartość zmierzona
float wart_nap; //zmienna na wartość zmierzonego napięcia

void setup() { //funkcja inicjalizacji
  lcd.begin(16, 2); //liczba kolumn i liczba linii LCD
  analogReference(DEFAULT); //napięcie odniesienia dla przetwornika A/C
  // (domyślnie 5V)
}

void loop() { //pętla główna programu
  lcd.clear(); //czyszczenie LCD
  lcd.setCursor(3, 0);
  lcd.print("Analog I/O"); //wyświetlenie na LCD napisu Analog I/O
  wart_pot = analogRead(A0); //pomiar napięcia
  wart_nap=(5.0*wart_pot)/1024.0; //konwersja liczby na napięcie
  lcd.setCursor(0, 1);
  lcd.print("U="); //wyświetlenie napisu U=
  lcd.print(wart_nap); //wyświetlenie napięcia z potencjometru
  lcd.print("V"); //wyświetlenie znaku V
  lcd.setCursor(8, 1);
  lcd.print("A/C="); //wyświetlenie napisu A/C=
  lcd.print(wart_pot); //wyświetlenie zmierzonej wartości przez A/C
  delay(300); //opóźnienie 300ms
} //koniec pętli głównej programu

```