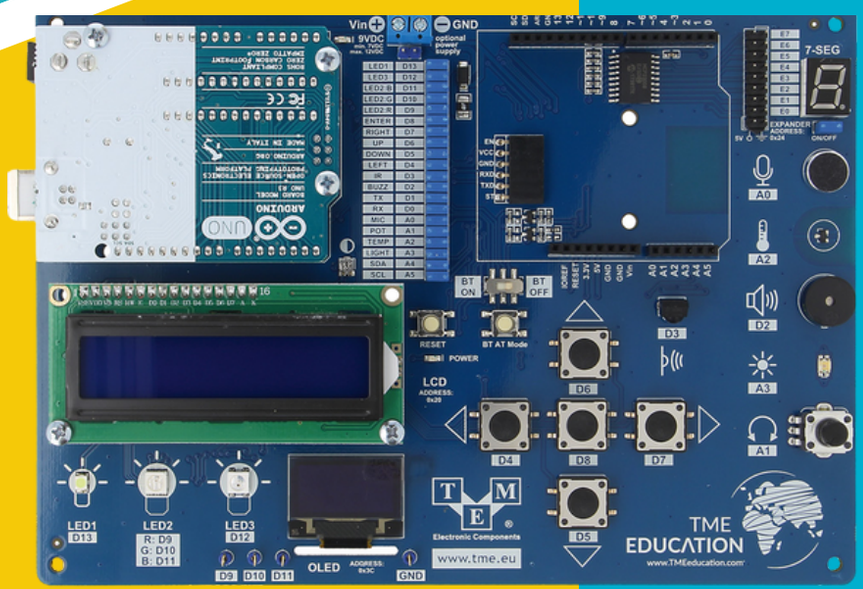
moje bambino®

MICHAŁ DURKIEWICZ

LEKCJE PROGRAMOWANIA MIKROKONTROLERA ARDUINO UNO

Z ZESTAWEM EDUKACYJNYM TME-EDU-ARD-2



PRODUKT DOSTĘPNY W MOJE BAMBINO

KOD PRODUKTU: 852500

ŁÓDŹ 2022

Michał Durkiewicz

Lekcje programowania mikrokontrolera Arduino Uno z zestawem edukacyjnym TME-EDU-ARD-2

Łódź • 2022

Spis treści

Lekcja 1

Pierwszy program z zestawem edukacyjnym TME-EDU-ARD-2, czyli budzimy Arduino UNO.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Zdobycie wiedzy na temat obsługi środowiska Arduino IDE. Zdobycie wiedzy na temat podstawowych instrukcji języka C++ i funkcji koniecznych do pracy z układem Arduino UNO. Ukształtowanie umiejętności kompilacji, wgrywania i testowania programu do zestawu edukacyjnego TME-EDU-ARD-2.

Efekty kształcenia

- Uczeń wie, na czym polega obsługa środowiska Arduino IDE.
- Uczeń umie napisać prosty program w języku C++ na platformę Arduino UNO.
- Uczeń zna podstawowe funkcje i instrukcje języka C++ na platformie Arduino UNO.
- Uczeń potrafi kompilować, wgrywać i testować program do zestawu edukacyjnego TME-EDU-ARD-2.
- Uczeń potrafi weryfikować poprawność działania swoich programów.

Wstęp

Niniejszy cykl lekcji wprowadzi Ciebie oraz Twoich uczniów w świat elektroniki i programowania w jak najpraktyczniejszej formie. Zastanawialiście się kiedyś, czy lekcje informatyki i realizacja podstawy programowej w zakresie algorytmiki i programowania musi odbywać się jedynie na komputerze? W tak mało praktycznej, nieciekawej formie?

W odpowiedzi na te pytania powstał poniższy cykl lekcji, z których możesz skorzystać w dowolnej kolejności, wybierając lekcje dostosowane do poziomu zaawansowania Twoich uczniów i poziomu, który sam chcesz osiągnąć z użyciem proponowanego zestawu.

Tematy lekcji podzieliliśmy wg następującego schematu.

Tematy 1–3 to podstawy programowania zestawu edukacyjnego TME-EDU-ARD-2. Tematy te wprowadzają w świat programowania Arduino UNO

Tematy 4–10 nauka programowania w języku C++ zestawu edukacyjnego TME-EDU-ARD-2 z uwzględnieniem wybranych zagadnień zawartych w podstawie programowej z informatyki dla szkoły podstawowej i ponadpodstawowej.

Tematy 11–15 nauka podstaw elektroniki i programowania obiektowego w oparciu o zestaw edukacyjny TME-EDU-ARD-2.

Wszystkie lekcje oparte są na zestawie edukacyjnym TME-EDU-ARD-2, którego popularność pozwala stwierdzić, że jest to najlepsze narzędzie do rozpoczęcia przygody z programowaniem i systemami mikroprocesorowymi.

Zestaw edukacyjny TME-EDU-ARD-2 jest to kompletne rozwiązanie, z którym bez dodatkowych komponentów możesz od razu rozpocząć pracę z uczniami. Zestaw oparty jest na jednym z najpopularniejszych na świecie sterowników Arduino UNO. A zintegrowane w zestawie peryferia dają szerokie spektrum możliwości do pracy nad poznaniem tajemnic informatyki i programowania.

Nieocenione w pracy z uczniami jest to, że cały zestaw edukacyjny jest bardzo solidnie zbudowany. Wszystkie komponenty są wlutowane w układ. Pogrubiona została płytka PCB. Pod płytką zostały umieszczone stopki na których stoi układ. Brak jest luźnych przewodów, które łączyłyby peryferia. Tak jak to jest w pracy z płytką stykową. Brak jest konieczności montowania układu przed każdą lekcją. Układ jest od razu gotowy do użycia. Wszystko to sprawia, że praca z układem jest przyjemna, bezpieczna i dająca ogromną satysfakcję z pracy w programowaniu systemu mikroprocesorowego jakim jest zestaw edukacyjny TME-EDU-ARD-2.

W zestawie TME-EDU-ARD-2 do naszej dyspozycji znajdziemy m. in.:

- Diodę świecącą LED (LED1 - D13).
- Kolorową diodę świecącą RGB (LED2 - D9, D10, D11).
- Sterowna cyfrowo, trójkolorowa (RGB) dioda świecąca WS2812 (LED3 - D12), do której szeregowo podłączono kolejne 4, identyczne diody umieszczone na spodzie płytki.
- Klawiatura składająca się z 5 przycisków monostabilnych rozmieszczonych w układzie przód (D6), tył (D5), lewo (D4), prawo (D7) oraz środek (D8).
- Czujniki analogowe:
  - Potencjometr (A1),
  - Czujnik światła (A3),
  - Termometr (A2),
  - Mikrofon (A0).
- Odbiornik podczerwieni (D3).
- Buzzer z generatorem (D2).
- Wyświetlacz tekstowy LCD 2x16 znaków, podłączony przez ekspander I2C (adres: 0x20).
- Wyświetlacz graficzny OLED z interfejsem I2C (adres: 0x3C).
- Wyświetlacz 7-segmentowy, podłączony przez ekspander I2C (adres: 0x24).

Przygotowanie komputera/pracowni do lekcji

Najważniejszym przed przystąpieniem do lekcji jest zainstalowanie na komputerach w pracowni środowiska do programowania układu ArduinoUNO. Niniejszy cykl lekcji oparty jest na darmowym oprogramowaniu Arduio IDE.

Oprogramowanie dostępne jest pod adresem:

`https://www.arduino.cc/en/software`

Kroki instalacji i konfiguracji oprogramowania Arduino IDE

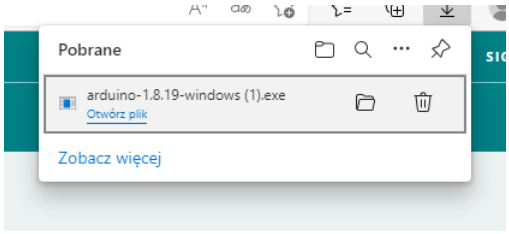
1. Pobranie właściwej wersji programu `https://www.arduino.cc/en/software`.
Zawsze korzystaj z najnowszej wersji oprogramowania. Pobierając Arduino IDE wybierz instalację zgodną z Twoim systemem operacyjnym.

Po wybraniu wersji programu otrzymasz prośbę o opcjonalne wsparcie dla twórców oprogramowania Arduino IDE.

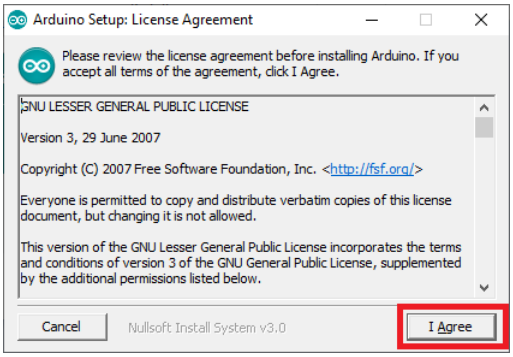
Jeśli nie chcesz przekazać opcjonalnego wsparcia kliknij **JUSTDOWNLOAD**.

W przypadku chęci udzielenia wsparcia finansowego dla twórców oprogramowania kliknij **CONTRIBUTE & DOWNLOAD**

2. Uruchomienie instalacji/ otwarcie pliku.

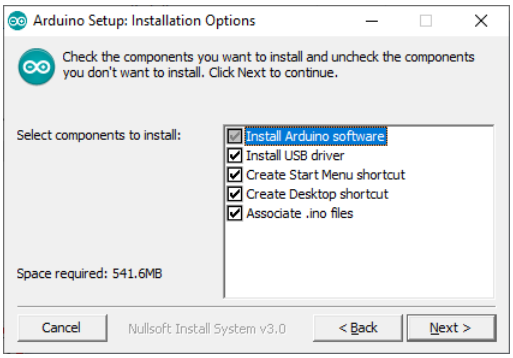


3. Akceptacja licencji Arduino IDE.
Oprogramowanie Arduino IDE oparte jest na licencji GNU. Tzn. Użytkownik może swobodnie i nieodpłatnie instalować, kopiować i udostępniać oprogramowanie Arduino IDE.

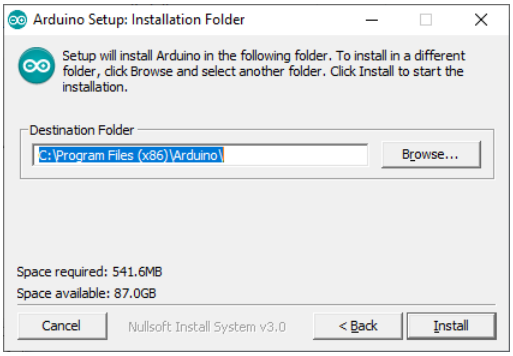


4. Wybranie komponentów do instalacji i konfiguracji.

- Install Arduino software (wymagane)
- Install USB driver (zalecane – jest to sterownik kontrolera USB wbudowanego w płytce Arduino UNO, bez niego nie ma możliwości wgrywania programów)
- Create Start Menu shortcut (zalecane)
- Create Desktop shortcut (opcjonalne)
- Associate .ino files (zalecane – pliki z rozszerzeniem .ino są domyślnym rozszerzeniem projektów zapisanych w środowisku Arduino IDE, zaznaczenie tej opcji ułatwi późniesze otwieranie zapisywanych projektów na dysku.

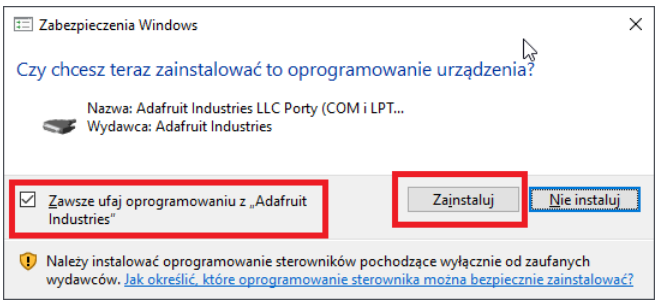


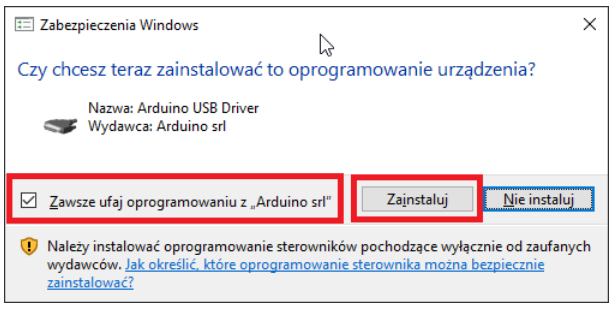
5. Wskazanie miejsca na dysku do instalacji.



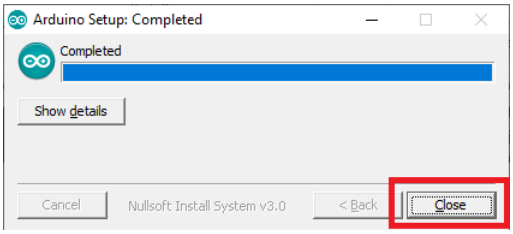
W trakcie instalacji możemy zostać poproszeni o potwierdzenie pobrania pliku instalacji z zaufanych źródeł. Zaznaczamy opcję: „Zawsze ufaj oprogramowaniu z „Adafruit Industries” ” oraz klikamy „Zainstaluj”.

Podobnie w kolejnych oknach zaznaczamy „Zawsze ufaj oprogramowaniu z „Arduino srl””, „Zawsze ufaj oprogramowaniu z „Arduino LLC” ” i również klikamy „Zainstaluj”.





6. Potwierdzenie kompletności instalacji.

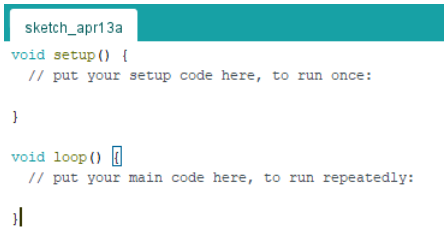


Przebieg zajęć

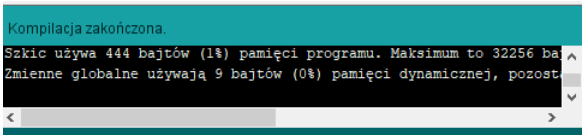
1. Uruchomienie przez nauczyciela i uczniów środowiska ArduinoIDE - domyślnie skrót do programu został utworzony w Menu Start i na Pulpicie.

2. Omówienie przez nauczyciela interfejsu programu Arduino IDE

Poniżej znajduje się widok pola edycji programu. W tej części okna znajduje się kod źródłowy naszego programu.



W dolnej części okna znajdują się informacje kompilatora. Ta część okna w momencie uruchomienia jest pusta. Po pierwszej kompilacji wypełni się danymi. Przykład:



Powyżej pola edycji znajduje się pasek narzędzi głównych Arduino IDE.

Opis opcji narzędzi głównych:

- opcja ”Zweryfikuj” - uruchamia kompilacje naszego programu. Proces kompilacji jest to działanie w którym kod naszego programu tłumaczony jest na kod wynikowy zrozumiały dla układu mikrokontrolera znajdującego się na płytce Arduino UNO.

- opcja ”Wgraj” - głównym zadaniem tej opcji jest przesłanie skompilowanego kodu wynikowego naszego programu do płytki Arduino UNO.

- opcja ”Nowy” - otwiera nowy plik projektu programu.

- opcja ”Otwórz” - pozwala na otwarcie wcześniej zapisanego na dysku twardym kodu źródłowego

- Opcja ”Zapisz” - pozwala zapisać projekt kodu źródłowego na dysku twardym. Uwaga: podczas zapisu w wybranej lokalizacji tworzony jest folder o nazwie zgodnej z podaną nazwą projektu. W tym folderze utworzony zostaje plik o domyślnym rozszerzeniu *.ino.

- Opcja ”Monitor portu szeregowego” - opcja pozwalająca na komunikację zestawu edukacyjnego z komputerem w trakcie działania programu w układzie Arduino UNO. Monitor portu szeregowego zostanie omówiony w kolejnych lekcjach.

3. Wykonanie wraz z nauczycielem ćwiczenia 1. Utworzenie pierwszego programu.

Przepisz i przetestuj poniższy program, który włączy diodę świecącą LED (D1 – pin 13).
Po jego wykonaniu przenalizuj działanie w oparciu o informacje od nauczyciela.

Kod źródłowy rozwiązania ćwiczenia:

```
void setup() {
  // Ustawienie pinu jako wyjście
  pinMode(13, OUTPUT);
  // ustawienie na wyjściu 13 stanu wysokiego, włączenie diody LED
  digitalWrite(13, HIGH);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Powyższy program przedstawia dwie najważniejsze funkcje, które muszą znaleźć się w programie działającym w układzie Arduino UNO. Jest to `void setup()` oraz `void loop()`. Funkcja `void setup()` w swoim bloku instrukcji będzie zawierać polecenia, które przez układ zostaną wykonane tylko raz. Wykonanie tych instrukcji nastąpi na początku, po uruchomieniu układu.

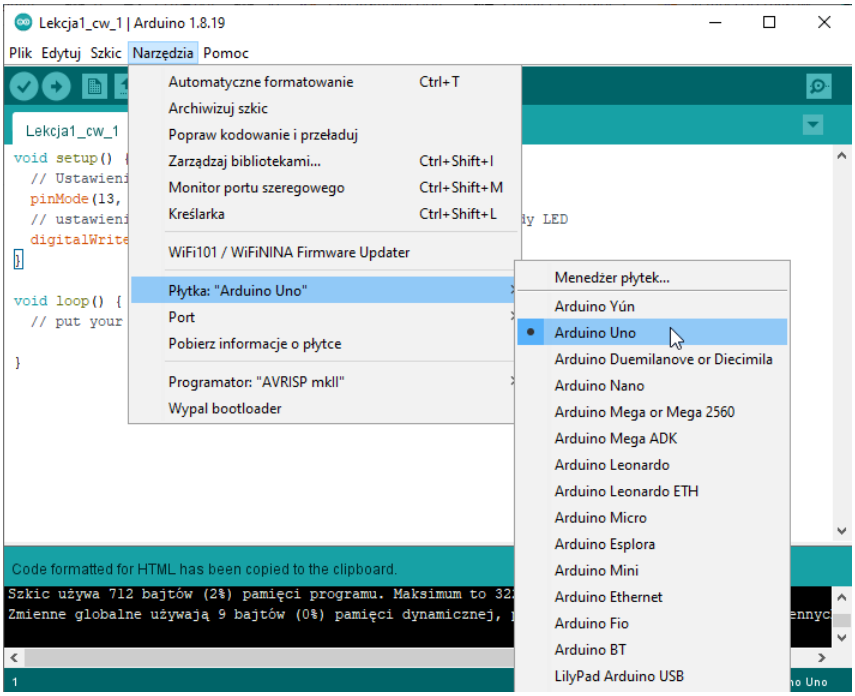
Funkcja `void loop()` będzie zawierać instrukcje wykonywane cyklicznie w pętli. Rozpoczęcie działania funkcji `void loop()` następuje bezpośrednio po wykonaniu instrukcji znajdujących się w funkcji `void setup()`.

W programie zastosowano również komentarze. Tzn. linie tekstu zapisane w programie źródłowym, które nie będą kompilowane. Komentarze mogą zawierać nasze opisy tego za co odpowiada dana instrukcja lub mogą „ukrywać" dla kompilatora fragmenty kodu, którego nie chcemy by był dołączony do kodu wynikowego. Komentarze jednolnijkowe symbolizujemy przez dwa znaki slash np. `// put your main code here, to run repeatedly:`, po których umieszczamy tekst komentarza.

W funkcji `void setup()` umieszczono instrukcję `pinMode(13, OUTPUT);`, która konfiguruje wyprowadzenie 13 z mikrokontrolera układu jako wyjście sygnału. Domyślnie układ na każdym z wyprowadzeń, może przyjąć różne funkcje. Możemy odczytywać sygnały lub je wyprowadzać. W tym przypadku właśnie do pinu 13 podłączona jest wyprowadzenie (anoda) diody LED. Jeśli chcemy ją włączyć to na wyprowadzeniu 13 powinno się pojawić napięcie 5V (stan wysoki). Właśnie za to odpowiada instrukcja `digitalWrite(13, HIGH);`. Stan ten (dioda włączona) będzie trwał aż do zaniku zasilania.

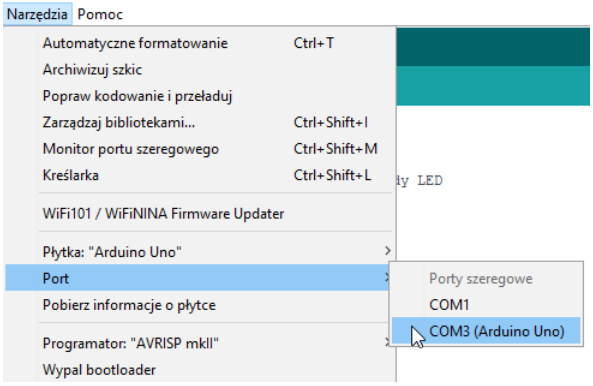
Wgrywanie programu do układu Arduino UNO

- wybieramy płytkę „Arduino UNO" jako płytkę na której pracujemy. Menu Narzędzia → Płytka: → Arduino UNO



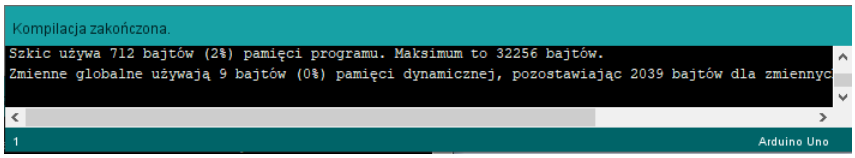
- podłączamy zestaw edukacyjny TME-EDU-ARD-2 do komputera. Do podłączenia układu służy kabel USB A-B dołączony do zestawu.

- wybieramy port COM pod którym system zainstalował nasz zestaw edukacyjny. Menu Narzędzia -< Port – COM3 (Arduino UNO) W prezentowanym przypadku płytka Arduino UNO znajduje się na porcie COM 3.

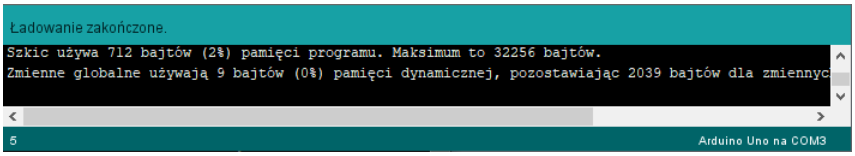


- za pomocą opcji „Zweryfikuj” kompilujemy program.

- sprawdzamy, czy kompilator w komunikatach kompilacji nie wskazał błędów. Poprawny proces kompilacji powinien zakończyć się komunikatem „Kompilacja zakończona”.



- Wgrywamy program za pomocą opcji „Wgraj”. Poprawny proces wgrywania powinien zakończyć sie komunikatem „Ładowanie zakończone”.



- Na układzie powinniśmy zobaczyć załączoną diodę LED D1.

4. Ćwiczenie 2. Włączenie na 1 sekundę diody LED.

Napisz program, który włączy na jedną sekundę diodę świecącą LED D1. Po tym czasie dioda zostanie wyłączona, aż do ponownego uruchomienia zestawu edukacyjnego.

Kod źródłowy rozwiązania ćwiczenia:

```
void setup() {
  // Ustawienie pinu jako wyjście
  pinMode(13, OUTPUT);
  // ustawienie na wyjściu 13 stanu wysokiego, włączenie diody LED
  digitalWrite(13, HIGH);
  // funkcja wstrzymująca program na 1 sekundę = 1000 milisekund
  delay(1000);
  // ustawienie na wyjściu 13 stanu niskiego, wyłączenie diody LED
  digitalWrite(13, LOW);
}

void loop() {
```

```
  // put your main code here, to run repeatedly:
}
```

W powyższym programie uzupełniono program o funkcję `delay(1000);`, która wstrzymuje działanie programu na 1000 milisekund. Po tym czasie następuje kontynuacja programu. Funkcja `digitalWrite(13, LOW);` ustawia wyprowadzenie 13 (diodę LED) w stan niski. Po tym następuje zakończenie programu. Cykl będzie powtarzał przy każdym uruchomieniu programu lub jego restarcie.

5. Ćwiczenie 3.

Napisz program, który będzie włączał na jedną sekundę diodę świecącą LED D1. Po czym nastąpi jedna sekunda przerwy w świeceniu diody LED i ponowne jej uruchomienie. Program powinien powtarzać włączanie i wyłączenie diody świecącej D1, aż do momentu, w którym odłączone zostanie zasilanie.

Kod źródłowy rozwiązania ćwiczenia:

```
void setup() {
  // Ustawienie pinu jako wyjście
  pinMode(13, OUTPUT);
}

void loop() {
   // ustawienie na wyjściu 13 stanu wysokiego, włączenie diody LED
  digitalWrite(13, HIGH);
  // wstrzymanie programu na 1000 milisekund
  delay(1000);
  // ustawienie na wyjściu 13 stanu niskiego, wyłączenie diody LED
  digitalWrite(13, LOW);
    // ponowne wstrzymanie programu na 1000 milisekund
  delay(1000);
}
```

W funkcji `void setup()` znajduje się jedynie konfiguracja pinu 13 (wyjście diody D1) jako wyjście sygnału.

Instrukcje odpowiedzialne za sekwencyjne włączanie/włączanie diody znajdują się w funkcji `void loop()`. W funkcji tej następuje kolejno: włączenie diody (`digitalWrite (13, HIGH);`), wstrzymanie programu na 1 sekundę (`delay (1000);`), wyłączenie diody (`digitalWrite (13, LOW);`) wstrzymanie programu na 1 sekundę (`delay (1000);`). Po wykonaniu tych czynności funkcja `void loop()` wykonuje się ponownie co sprawia, że dioda LED D1 ponownie się włącza i wyłącza.

6. Przeprowadzenie quizu z lekcji. Poprawne odpowiedzi zostały zaznaczone.

1. Układ na którym oparty jest zestaw edukacyjny TME-EDU-ARD-2 to:
   a) Arduino IDE
   b) PlatformIO
   c) Arduino UNO
   d) Atmega328

2. Funkcja w programie, która wykonuje się jedynie raz po uruchomieniu i jest odpowiedzialna za konfigurację układu to:
   a) `void loop()`
   b) `void setup()`
   c) `void delay()`
   d) `digitalWrite(13, HIGH);`
3. Wskaż funkcję, która konfiguruje wyprowadzenie diody D1 jako wyjście:
   a) `digitalWrite(13, HIGH);`
   b) `digitalWrite(13, LOW);`
   c) `pinMode(13, OUTPUT);`
   d) `delay(1000);`
4. Wskaż funkcję, której użycie spowoduje wstrzymanie działania programu na 1 sekundę:
   a) `delay(1);`
   b) `delay(1/60);`
   c) `delay(100);`
   d) `delay(1000);`
5. Wskaż funkcję, która na wyprowadzeniu pinu diody LED ustawi stan wysoki (włączy diodę LED):
   a) `digitalWrite(13, HIGH);`
   b) `digitalWrite(13, LOW);`
   c) `pinMode(13, OUTPUT);`
   d) `delay(1000);`

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Przepisz i przetestuj poniższy program, który włączy diodę świecącą LED (D1 – pin 13).
Po jego wykonaniu przenalizuj działanie w oparciu o informacje od nauczyciela.

Kod źródłowy rozwiązania ćwiczenia:

```
void setup() {
  // Ustawienie pinu jako wyjście
  pinMode(13, OUTPUT);
  // ustawienie na wyjściu 13 stanu wysokiego, włączenie diody LED
  digitalWrite(13, HIGH);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Ćwiczenie 2

Napisz program, który włączy na jedną sekundę diodę świecącą LED D1. Po tym czasie dioda zostanie wyłączona, aż do ponownego uruchomienia zestawu edukacyjnego.

Kod źródłowy rozwiązania ćwiczenia:

```
void setup() {
  // Ustawienie pinu jako wyjście
  pinMode(13, OUTPUT);
  // ustawienie na wyjściu 13 stanu wysokiego, włączenie diody LED
  digitalWrite(13, HIGH);
  // funkcja wstrzymująca program na 1 sekundę = 1000 milisekund
  delay(1000);
  // ustawienie na wyjściu 13 stanu niskiego, wyłączenie diody LED
  digitalWrite(13, LOW);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Ćwiczenie 3

Napisz program, który będzie włączał na jedną sekundę diodę świecącą LED D1. Po czym nastąpi jedna sekunda przerwy w świeceniu diody LED i ponowne jej uruchomienie. Program powinien powtarzać włączanie i wyłączenie diody świecącej D1, aż do momentu, w którym odłączone zostanie zasilanie.

Quiz

1. Układ na którym oparty jest zestaw edukacyjny TME-EDU-ARD-2 to:
   a) Arduino IDE
   b) PlatformIO
   c) Arduino UNO
   d) Atmega328
2. Funkcja w programie, która wykonuje się jedynie raz po uruchomieniu i jest odpowiedzialna za konfigurację układu to:
   a) `void loop()`
   b) `void setup()`
   c) `void delay()`
   d) `digitalWrite(13, HIGH);`
3. Wskaż funkcję, która konfiguruje wyprowadzenie diody D1 jako wyjście:
   a) `digitalWrite(13, HIGH);`
   b) `digitalWrite(13, LOW);`
   c) `pinMode(13, OUTPUT);`
   d) `delay(1000);`
4. Wskaż funkcję, której użycie spowoduje wstrzymanie działania programu na 1 sekundę:
   a) `delay(1);`
   b) `delay(1/60);`
   c) `delay(100);`
   d) `delay(1000);`
5. Wskaż funkcję, która na wyprowadzeniu pinu diody LED ustawi stan wysoki (włączy diodę LED):
   a) `digitalWrite(13, HIGH);`
   b) `digitalWrite(13, LOW);`
   c) `pinMode(13, OUTPUT);`
   d) `delay(1000);`

Lekcja 2

Sterujemy przyciskami w zestawie edukacyjnym TME-EDU-ARD-2.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Zdobycie wiedzy na temat możliwości wykorzystania przycisków monostabilnych wbudowanych w płytkę zestawu edukacyjnego TME-EDU-ARD-2. Nabycie umiejętności stosowania instrukcji warunkowej jeżeli. Nabycie wiedzy nt. stosowania zmiennych całkowitych.

Efekty kształcenia

- Uczeń wie, co to jest przycisk monostabilny.
- Uczeń potrafi napisać program odczytujący stan przycisku monostabilnego.
- Uczeń zna pojęcie zmiennej.
- Uczeń potrafi stosować zmienne całkowite (integer).
- Uczeń potrafi stosować instrukcję warunkową jeżeli (`if`).

Wstęp

Lekcja ta jest kontynuacją tematu 1 lekcji naszego cyklu. W lekcji tej Ty i Twoi uczniowie poznacie, czym są przyciski monostabilne, i jak możemy je wykorzystywać praktycznie w naszych programach.

Ćwiczenia, które proponujemy w trakcie lekcji, nie tylko wskażą jak odczytywać, czy przyciski monostabilne wbudowane w nasz zestaw są wciśnięte czy też nie. Pokażą one również w praktycznym wymiarze czym jest i jak możemy zastosować instrukcję warunkową (`if`). Ponadto będziemy za pomocą przycisków zmieniać wartość przechowywaną w zmiennej całkowitej typu integer.

Przebieg zajęć

1. Przedstawienie celów lekcji przez nauczyciela.

2. Powtórzenie z uczniami, jak działały następujące funkcje:

```
1. void setup()
2. void loop()
3. pinMode(13, OUTPUT);
4. digitalWrite(13, HIGH);
```

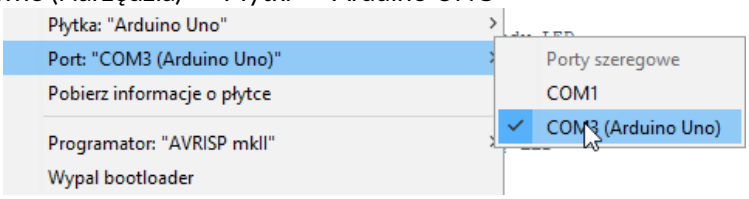
5. `digitalWrite(13, HIGH);`

3. Powtórzenie z uczniami, w jaki sposób tworzymy nowy projekt.

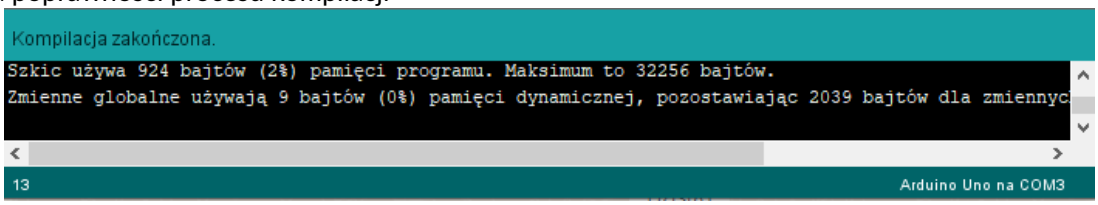
4. Powtórzenie z uczniami, co musimy sprawdzić i wykonać, by wgrać nasz program do płytki Arduino UNO.

1. Ustawienie właściwej płytki. Menu główne (Narzędzia) → Płytki → Arduino UNO

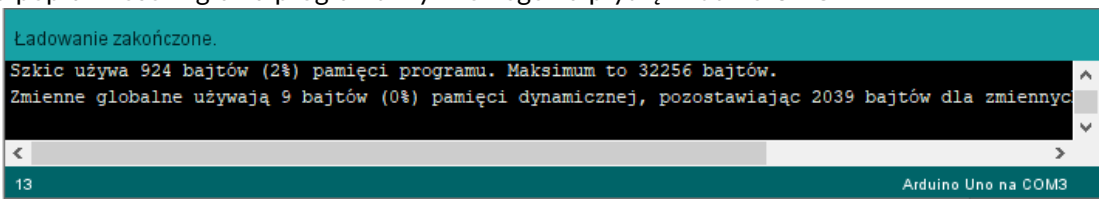
2. Ustawienie właściwego portu. Menu główne (Narzędzia) → Płytki → Arduino UNO



3. Kompilacja programu przyciskiem  .
4. Weryfikacja poprawności procesu kompilacji



5. Wgranie programu wynikowego przyciskiem 
6. Weryfikacja poprawności wgrania programu wynikowego na płytkę Arduino UNO



5. Omówienie z uczniami instrukcji warunkowej (`if`).

Czym jest instrukcja warunkowa? Instrukcja warunkowa w programowaniu służy do realizacji wybranej sekwencji instrukcji w zależności od tego, czy zostało spełnione określone kryterium.

Kryterium nazywamy warunek logiczny, który jeśli jest spełniony (prawda) pozwala na wykonanie określonej sekwencji instrukcji. Możliwe jest wprowadzenie sekwencji instrukcji alternatywnych, które zostaną wykonane w przypadku niespełnienia warunku (`else`).

Składnia instrukcji warunkowej bez alternatywy:

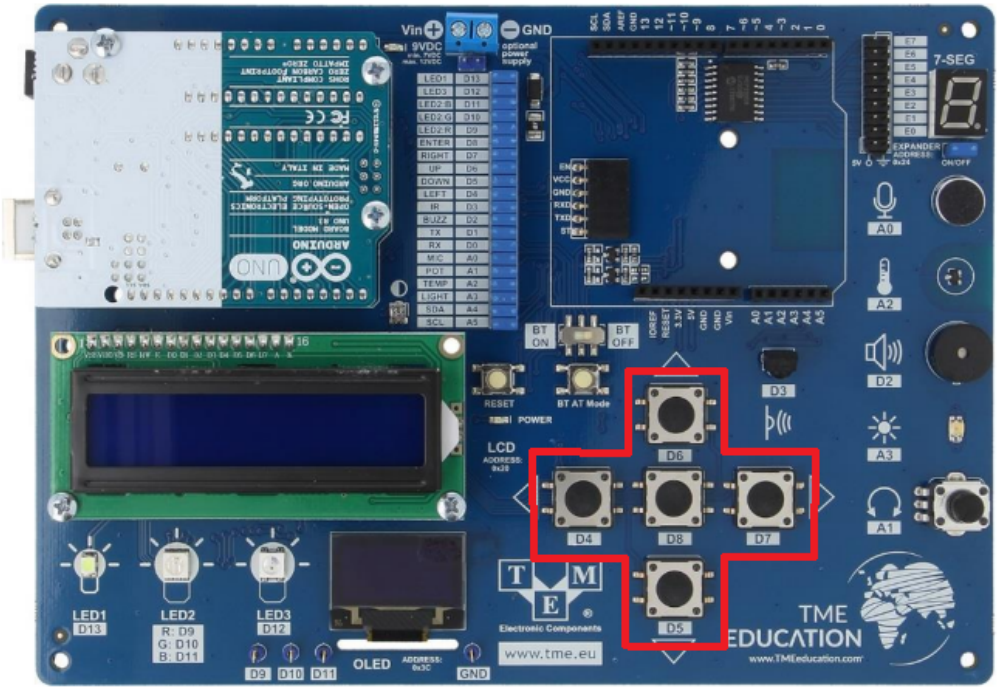
```
if( ⟨warunek logiczny⟩ )
{
    ⟨sekwencja instrukcji wykonywana, gdy warunek logiczny jest spełniony⟩
}
```

Składnia instrukcji warunkowej z alternatywą:

```
if( ⟨warunek logiczny⟩ )
{
    ⟨sekwencja instrukcji wykonywana, gdy warunek logiczny jest spełniony⟩
}
else
{
    ⟨sekwencja instrukcji wykonywana, gdy warunek logiczny nie jest spełniony⟩
}
```

6. Omówienie z uczniami czym są i jak działają przyciski monostabilne wbudowane w zestaw edukacyjny TME-EDU-ARD-2.

W proponowanym zestawie edukacyjnym zostało wbudowanych 5 przycisków monostabilnych. Znajdują się one w centralnej części układu, i ułożone są między sobą w układzie krzyżowym.



Przyciski monostabilne działają w ten sposób, że w momencie nacisku na przycisk powstaje zwarcie (połączenie) w obwodzie z przyciskiem i wyprowadzenia przycisku („nóżki") są ze sobą połączone.

W zestawie edukacyjnym TME-EDU-ARD-2 kolejne przyciski podłączone są przez bufory od-

wracające. Z jednej strony podłączone są do wyprowadzeń D4, D5, D6, D7, D8 w układzie Arduino UNO, a z drugiej do masy.

Obecność buforu odwracającego powoduje, że jeśli przycisk zostanie naciśnięty, to na wyprowadzeniu, do którego jest podłączony, pojawi się stan wysoki.

Uwaga! Stan wysoki będzie się utrzymywał tak długo, jak długo przycisk będzie wciśnięty. Po zwolnieniu przycisku stan zmienia się na niski i takim pozostaje do następnego wciśnięcia.

7. Omówienie i wykonanie przez nauczyciela ćwiczenia 1.

Wykonaj program, który włączy diodę świecącą LED D1 w chwili, kiedy przycisk środkowy D8 jest wciśnięty. Z chwilą puszczenia przycisku program powinien wyłączyć diodę LED D1.

Kod źródłowy do ćwiczenia:

```
// definicja stałych LED1 SW_CENTER jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przycisk monostabilny środkowy
#define LED1 13
#define SW_CENTER 8

// instrukcje konfigurujące wyprowadzenia układu Arduino
void setup() {
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // instrukcja warunkowa sprawdzająca, czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH) {
    // sekwencja instrukcji wykonywana, jeśli przycisk jest wciśnięty
    digitalWrite(LED1, HIGH);
  } else {
    digitalWrite(LED1, LOW); // sekwencja instrukcji wykonywana, jeśli nie jest wciśnięty
  }
}
```

W programie zastosowano stałe `LED1` oraz `SW_CENTER`. Stałe w programowaniu C++ pozwalają zapamiętać pod opisami słownymi dane (liczbowe, logiczne, znakowe, łańcuchowe). Takie przypisanie określeń słownych dla liczb (i innych danych, jak okaże się później), których używamy wielokrotnie, zwiększa czytelność programu. Wskazuje znaczenie danej liczby w danym miejscu programu. Ułatwia wprowadzanie ewentualnych zmian w programie.

Przy programowaniu zestawu edukacyjnego z Arduino UNO niewygodne i trudne byłoby pamiętać numery wszystkich peryferii: wyprowadzeń przycisków, diod LED i innych. Dlatego w naszych programach warto dla określenia pinu (wyprowadzenia) danego podzespołu zdefiniować stałą.

Stałe definiujemy za pomocą polecenia `#define`, a następnie wpisujemy nazwę i wartość (rozdzielając tylko spacjami). Umownie w nazwach stałych używamy wielkich liter, cyfr, znaków podkreślenia „_”.
WAŻNE: W nazwach stałych nie używamy spacji!

Przykład:

```
#define LED1 13
```

Dzięki definicji stałych tam, gdzie w programie wskazujemy pin diody D1, posługujemy się nazwą (stałą) `LED1`. Tam, gdzie wskazujemy na przycisk środkowy, posługujemy się stałą `SW_CENTER`.

W funkcji `void setup()` następuje konfiguracja pinów: wyjściowego dla diody LED D1 oraz wejściowego dla przycisku.

W funkcji `void loop()` cyklicznie następuje sprawdzenie, czy przycisk środkowy jest wciśnięty (czy na pinie, do którego jest podłączony, jest stan wysoki). Odpowiada za to instrukcja warunkowa, w której warunek został sformułowany następująco:

```
digitalRead(SW_CENTER) == HIGH,
```

co czytamy: „Czy na pinie przycisku środkowego (`SW_CENTER`) jest stan wysoki (jest wciśnięty przycisk)?”.

Jeśli przycisk jest wciśnięty (warunek spełniony), to następuje włączenie diody LED D1.

W przeciwnym przypadku dioda LED D1 zostaje wyłączona.

8. Samodzielne wykonanie przez uczniów ćwiczenia 2

Wykonaj program, który będzie włączał i wyłączał co 1 sekundę diodę LED D1 w chwili, kiedy przycisk środkowy D8 jest wciśnięty. Z chwilą puszczenia przycisku program powinien wyłączyć diodę LED D1.

Kod źródłowy do ćwiczenia:

```
// definicja stałych LED1 i SW_CENTER jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przycisk monostabilny środkowy
#define LED1 13
#define SW_CENTER 8

// instrukcje konfigurujące wyprowadzenia układu Arduino
void setup() {
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // instrukcja warunkowa sprawdzająca, czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH) {
```

```
    // sekwencja instrukcji wykonywana, jeśli przycisk jest  wciśnięty
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
  } else {
    digitalWrite(LED1, LOW); // sekwencja instrukcji wykonywana, jeśli przycisk nie jest wciśnięty
  }
}
```

W programie w przypadku wciśnięcia przycisku i włączenia diody LED D1, zastosowano funkcję `delay(1000);`, która wstrzymuje działanie programu przez sekundę. Następnie wyłączona zostaje dioda LED D1 i ponownie program zostaje wstrzymany przez 1 sekundę. Po wykonaniu tych instrukcji funkcja `void loop()` wykonuje się ponownie i ponownie sprawdza czy przycisk jest wciśnięty. Taka konstrukcja pozwala nam zbudować program spełniający wymagania ćwiczenia.

Uwaga 1. Po wykryciu, że przycisk jest wciśnięty, program musi wykonać pełny cykl włączenia i wyłączenia diody LED. Oznacza to, że jeśli nagle na początku świecenia diody przycisk zostanie wyłączony, to i tak dioda LED będzie świecić do pełnej sekundy od włączenia.

9. Samodzielne wykonanie przez uczniów ćwiczenia 3.

Wykonaj program, w którym:

- Przytrzymanie wciśniętego przycisku lewego D4 spowoduje, że program będzie włączał i wyłączał co 0,25 sekundy diodę LED D1.
- Przytrzymanie wciśniętego przycisku środkowego D8 spowoduje, że program będzie włączał i wyłączał co 0,5 sekundy diodę LED D1.
- Przytrzymanie wciśniętego przycisku prawego D7 spowoduje, że program będzie włączał i wyłączał co 1 sekundę diodę LED D1.
- Wciśnięcie przycisku górnego D6 spowoduje, że program włączy diodę LED D1 na stałe.
- Wciśnięcie przycisku dolnego D5 spowoduje, że program wyłączy diodę LED D1.

Kod źródłowy do ćwiczenia:

```
// definicja stałych jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przyciski monostabilne
#define LED1 13
#define SW_CENTER 8
#define SW_RIGHT 7
#define SW_LEFT 4
#define SW_UP 6
#define SW_DOWN 5
```

```
// instrukcje konfigurujące wyprowadzenia układu Arduino
void setup() {
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_LEFT, INPUT);
  pinMode(SW_CENTER, INPUT);
  pinMode(SW_RIGHT, INPUT);
  pinMode(SW_UP, INPUT);
  pinMode(SW_DOWN, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  if (digitalRead(SW_LEFT) == HIGH) {
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 0,25 sekundy
    delay(250);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 0,25 sekundy
    delay(250);
  }

  if (digitalRead(SW_CENTER) == HIGH) {
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 0,5 sekundy
    delay(500);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 0,5 sekundy
    delay(500);
  }
  if (digitalRead(SW_RIGHT) == HIGH) {
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
  }

  if (digitalRead(SW_UP) == HIGH) {
```

```
    // włączenie diody
    digitalWrite(LED1, HIGH);
  }
  if (digitalRead(SW_DOWN) == HIGH) {
    // wyłączenie diody
    digitalWrite(LED1, LOW);
  }
}
```

W programie wykorzystujemy wszystkie 5 przycisków monostabilnych. Dlatego na początku programu dokonaliśmy definicji 6 stałych, do których przyporządkowaliśmy piny, na których wyprowadzona jest dioda led D1 oraz kolejne 5 przycisków monostabilnych.

W funkcji `void setup()` musieliśmy skonfigurować piny przycisków jako piny wejściowe w układzie.

W funkcji `void loop()` dla każdego przycisku została przygotowana instrukcja warunkowa sprawdzająca, czy dany przycisk jest wciśnięty. Dla przycisków `SW_LEFT`, `SW_CENTER` i `SW_RIGHT` wykonujemy instrukcje podobne jak w ćwiczeniu 2, z tą różnicą, że dla każdego przycisku ustalamy inne czasy przerw w funkcji `delay()`. Pamiętamy, że funkcja `delay()` przyjmuje jako parametr wartość wyrażoną w milisekundach.
Dla przycisku `SW_UP` w instrukcji warunkowej zostało jedynie zapisane ustawienie stanu wysokiego na diodzie LED (włączenie diody). Natomiast dla przycisku `SW_DOWN` w instrukcji warunkowej zapisaliśmy ustawienie stanu niskiego na diodzie LED (wyłączenie diody).

10. Omówienie przez nauczyciela pojęcia zmiennej. Omówienie deklaracji zmiennej całkowitej.

Zmienna, podobnie jak stała, pozwala nam pod nazwą opisową zapamiętać dane liczbowe, znakowe, logiczne lub słowne. Zmienna jest swoistym „pudełkiem", które na „wieczku" zostało przez nas opisane umówionym słowem bądź połączeniem liter i cyfr. Do tego pudełka możemy wkładać dane (np. liczby), a jeśli chcemy ich użyć w programie – wystarczy, że wpiszemy nazwę naszej zmiennej czyli nazwę z „wieczka" naszego „pudełka". *zmienna*

Zmienne tym różnią się od stałych, że mogą być modyfikowane w trakcie działania programu. Np. do przechowania wyniku działania arytmetycznego.

Każda zmienna przed jej pierwszym użyciem w programie musi zostać zadeklarowana. Deklaracja zmiennej odbywa się przez podanie typu i nazwy.

Nazwy zmiennych powinny rozpoczynać się od małych liter, mogą zawierać cyfry oraz znak podkreślenia „_".
Nazwy zmiennych nie mogą zawierać spacji.

Typ zmiennej to rodzaj danych, jakie można w zmiennej przechowywać. Np., czy dana ma być liczbą całkowitą? Liczbą rzeczywistą? Znakiem? Słowem (tzw. łańcuchem znaków)?

W naszym pierwszym programie, który używa zmiennych, wykorzystamy zmienne całkowite. Do ćwiczeń z zestawem edukacyjnym TME-EDU-ARD-2 optymalny będzie typ liczb całkowitych integer (`int`). Deklaracja zmiennej o nazwie `liczba` tego typu wygląda następująco:

```
int liczba;
```

W zapisie tym `int` oznacza typ całkowity integer, a słowo liczba jest nazwą naszej zmiennej.

A tak wygląda nadanie powyższej zmiennej wartości 10, czyli „włożenie liczby całkowitej 10 do pudełka” `liczba` :

```
liczba=10;
```

Operacja nadania zmiennej wartości nazywa się **przypisaniem**. ***przypisanie***

Oto przykładowe działania na naszej zmiennej:

```
liczba= 5+10;
liczba= liczba +5
```

Pierwsze z powyższych działań przypisze do zmiennej `liczba` wartość 15.

Drugie z działań do wartości, jaką posiada aktualnie zmienna `liczba`, doda wartość 5 i wynik tego dodawania przypisze jako nową wartość zmiennej `liczba`.

11. Wykonanie wspólnie z uczniami ćwiczenia 4

Wykonaj program, w którym wciśnięcie przycisku środkowego D8 spowoduje naprzemienne cykliczne włączanie i wyłączanie diody LED D1 co 1 sekundę. Program będzie włączał i wyłączał diodę LED D1 aż do ponownego wciśnięcia przycisku D8 przez użytkownika.

Kod źródłowy do ćwiczenia:

```
// definicja stałych LED1 i SW_CENTER jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przycisk monostabilny środkowy
#define LED1 13
#define SW_CENTER 8

// deklaracja zmiennej całkowitej onoff,
// zmienna onoff będzie pamiętać stan włączenia diody.
// Jeśli 1 to dioda włączona, 0 to wyłączona.
int onoff;

// instrukcje konfigurujące wyprowadzenia układu Arduino:
void setup() {
  // zapisanie w zmiennej onoff informacji, że dioda jest wyłączona
  onoff=0;
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // instrukcja warunkowa sprawdzająca, czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH) {
    // sekwencja instrukcji wykonywana, jeśli przycisk wciśnięty
    // w tym przypadku musimy sprawdzić, czy aktualnie dioda jest włączona czy wyłączona
```

```
    // i zmienić ustawienie zmiennej onoff z 0 na 1 lub 1 na 0
    if(onoff==0)
      onoff=1;
    else
      onoff=0;
  }
  // jeżeli wartość zmiennej onoff mówi,
  // że diodę jest włączona (onoff==1), to wykonujemy miganie diodą
  if(onoff==1)
  {    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
  }

  else
  {
    // jeżeli na zmiennej onoff jest wartość 0, to wyłączamy diodę i czekamy 1 sekundę
    digitalWrite(LED1, LOW); // sekwencja instrukcji wykonywana, gdy przycisk nie jest wciśnięty
    delay(1000);
  }

}
```

W programie zastosowano zmienną `onoff` typu całkowitego integer. Zmienna ta w programie pełni rolę informacyjną: przechowywana jest w niej informacja o tym, czy miganie diody LED jest włączone, czy wyłączone. Wykonanie deklaracji powyżej funkcji `void setup()` oraz `void loop()` oznacza, że zmienna ta jest **widoczna** w każdej z nich, czyli że każda z tych funkcji może odczytywać i zmieniać wartość zmiennej `onoff`. Taka deklaracja (przed funkcjami – na początku programu) nosi nazwę deklaracji globalnej

widoczność zmiennej

W funkcji `void setup()` wykonujemy ustawienie wartości zmiennej onoff na 0. Czyli zapisujemy informację, że dioda nie mruga.

Funkcja `void loop()` zawiera instrukcję warunkową sprawdzającą, czy wciśnięty jest przycisk „środkowy" D8. Jeśli przycisk jest wciśnięty to w tej sytuacji używamy instrukcji warunkowej, która sprawdza stan zmiennej onoff (czy dioda aktualnie miga). Jeśli onoff=0 (dioda LED nie miga) to zmieniamy onoff na 1. Jeśli onoff=1 (dioda LED miga) to zmieniamy onoff na 0.

Jeśli przycisk jest wyłączony nie musimy nic zmieniać bo wtedy utrzymujemy dotychczasowy stan migania diody.

Poniżej warunku sprawdzającego wciśnięcie przycisku znajduje się instrukcja warunkowa sprawdzająca ustawienie wartości zmiennej onoff. Jeśli onoff jest równe 1 tzn. że program ma mrugać diodą (włączać diodę, czekać 1 sekundę, wyłączać diodę, czekać jedną sekundę). Jeśli onoff

jest równe 0 to wtedy musimy wyłączyć diodę i odczekać 1 sekundę do ponownego wykonania instrukcji `void loop()`.

Powyższe wstrzymanie programu jest konieczne, gdyż program mógłby mylnie interpretować lekkie przytrzymanie przycisku środkowego D8 jako ponowne jego wciśnięcie. Skutkowałoby to ponownym rozpoczęciem mrugania diody LED.

12. Przeprowadzenie quizu podsumowującego lekcję.

1. Wskaż zdanie nieprawdziwe
   a) Pinu przycisku monostabilnego nie trzeba konfigurować na początku programu
   b) Przycisk monostabilny po przyciśnięciu pozostaje włączony do ponownego wciśnięcia
   c) Na pinie przycisku monostabilnego w czasie, gdy jest on wyłączony, utrzymywany jest poziom niski (`LOW`)
   d) Na pinie przycisku monostabilnego w czasie, gdy jest on włączony, utrzymywany jest poziom wysoki (`HIGH`)
2. Która deklaracja stałej jest poprawna?
   a) `#define SW_CENTER 8`
   b) `#define SW_CENTER=8`
   c) `#define sw_center 8`
   d) `#define sw_center=8`
3. Który z poniższych napisów jest poprawnym warunkiem (testem) instrukcji `if` dającym informację o tym, czy przycisk lewy został wciśnięty?
   a) `if (digitalRead(SW_LEFT) = HIGH)`
   b) `if (digitalRead(SW_LEFT) == HIGH)`
   c) `if (digitalRead(SW_LEFT) = LOW)`
   d) `if (digitalRead(SW_LEFT) == LOW)`
4. Wskaż deklarację zmiennej całkowitej (typu `int`):
   a) `int liczba1;`
   b) `liczba1 int;`
   c) `int liczba 1;`
   d) `int LICZBA 1;`
5. Który z poniższych tekstów nie jest operacją na zmiennych w języku C++?
   a) `liczba=0;`
   b) `liczba=5+liczba;`
   c) `5+liczba=liczba;`
   d) `liczba=2*liczba;`

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Wykonaj program, który włączy diodę świecącą LED D1 w chwili, kiedy przycisk środkowy D8 jest wciśnięty. Z chwilą puszczenia przycisku program powinien wyłączyć diodę LED D1.

Kod źródłowy do ćwiczenia:

```
// definicja stałych LED1 SW_CENTER jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przycisk monostabilny środkowy
#define LED1 13
#define SW_CENTER 8

// instrukcje konfigurujące wyprowadzenia układu Arduino
void setup() {
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // instrukcja warunkowa sprawdzająca, czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH) {
    // sekwencja instrukcji wykonywana, jeśli przycisk jest wciśnięty
    digitalWrite(LED1, HIGH);
  } else {
    digitalWrite(LED1, LOW); // sekwencja instrukcji wykonywana, jeśli nie jest wciśnięty
  }
}
```

Ćwiczenie 2

Wykonaj program, który będzie włączał i wyłączał co 1 sekundę diodę LED D1 w chwili, kiedy przycisk środkowy D8 jest wciśnięty. Z chwilą puszczenia przycisku program powinien wyłączyć diodę LED D1.

Kod źródłowy do ćwiczenia:

```
// definicja stałych LED1 i SW_CENTER jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przycisk monostabilny środkowy
```

```
#define LED1 13
#define SW_CENTER 8

// instrukcje konfigurujące wyprowadzenia układu Arduino
void setup() {
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // instrukcja warunkowa sprawdzająca, czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH) {
    // sekwencja instrukcji wykonywana, jeśli przycisk jest wciśnięty
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
  } else {
    digitalWrite(LED1, LOW); // sekwencja instrukcji wykonywana, jeśli przycisk nie jest wciśnięty
  }
}
```

Ćwiczenie 3

Wykonaj program, w którym:

- Przytrzymanie wciśniętego przycisku lewego D4 spowoduje, że program będzie włączał i wyłączał co 0,25 sekundy diodę LED D1.
- Przytrzymanie wciśniętego przycisku środkowego D8 spowoduje, że program będzie włączał i wyłączał co 0,5 sekundy diodę LED D1.
- Przytrzymanie wciśniętego przycisku prawego D7 spowoduje, że program będzie włączał i wyłączał co 1 sekundę diodę LED D1.
- Wciśnięcie przycisku górnego D6 spowoduje, że program włączy diodę LED D1 na stałe.
- Wciśnięcie przycisku dolnego D5 spowoduje, że program wyłączy diodę LED D1.

Kod źródłowy do ćwiczenia:

```
// definicja stałych jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przyciski monostabilne
#define LED1 13
```

```
#define SW_CENTER 8
#define SW_RIGHT 7
#define SW_LEFT 4
#define SW_UP 6
#define SW_DOWN 5

// instrukcje konfigurujące wyprowadzenia układu Arduino
void setup() {
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_LEFT, INPUT);
  pinMode(SW_CENTER, INPUT);
  pinMode(SW_RIGHT, INPUT);
  pinMode(SW_UP, INPUT);
  pinMode(SW_DOWN, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  if (digitalRead(SW_LEFT) == HIGH) {
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 0,25 sekundy
    delay(250);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 0,25 sekundy
    delay(250);
  }

  if (digitalRead(SW_CENTER) == HIGH) {
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 0,5 sekundy
    delay(500);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 0,5 sekundy
    delay(500);
  }
  if (digitalRead(SW_RIGHT) == HIGH) {
    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
    // wyłączenie diody
```

```
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
  }

  if (digitalRead(SW_UP) == HIGH) {
    // włączenie diody
    digitalWrite(LED1, HIGH);
  }
  if (digitalRead(SW_DOWN) == HIGH) {
    // wyłączenie diody
    digitalWrite(LED1, LOW);
  }
}
```

Ćwiczenie 4

Wykonaj program, w którym wciśnięcie przycisku środkowego D8 spowoduje naprzemienne cykliczne włączanie i wyłączanie diody LED D1 co 1 sekundę. Program będzie włączał i wyłączał diodę LED D1 aż do ponownego wciśnięcia przycisku D8 przez użytkownika.

Kod źródłowy do ćwiczenia:

```
// definicja stałych LED1 i SW_CENTER jako numerów pinów,
// pod którymi podłączona jest dioda LED D1 oraz przycisk monostabilny środkowy
#define LED1 13
#define SW_CENTER 8

// deklaracja zmiennej całkowitej onoff,
// zmienna onoff będzie pamiętać stan włączenia diody.
// Jeśli 1 to dioda włączona, 0 to wyłączona.
int onoff;

// instrukcje konfigurujące wyprowadzenia układu Arduino:
void setup() {
  // zapisanie w zmiennej onoff informacji, że dioda jest wyłączona
  onoff=0;
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// instrukcje z funkcji loop wykonują się cały czas (w pętli)
void loop() {
  // instrukcja warunkowa sprawdzająca, czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH) {
    // sekwencja instrukcji wykonywana, jeśli przycisk wciśnięty
```

```
    // w tym przypadku musimy sprawdzić, czy aktualnie dioda jest włączona czy wyłączona
    // i zmienić ustawienie zmiennej onoff z 0 na 1 lub 1 na 0
    if(onoff==0)
      onoff=1;
    else
      onoff=0;
  }
  // jeżeli wartość zmiennej onoff mówi,
  // że diodę jest włączona (onoff==1), to wykonujemy miganie diodą
  if(onoff==1)
  {    // włączenie diody
    digitalWrite(LED1, HIGH);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
    // wyłączenie diody
    digitalWrite(LED1, LOW);
    // wstrzymanie programu na 1 sekundę
    delay(1000);
  }

  else
  {
    // jeżeli na zmiennej onoff jest wartość 0, to wyłączamy diodę i czekamy 1 sekundę
    digitalWrite(LED1, LOW); // sekwencja instrukcji wykonywana, gdy przycisk nie jest wciśnięty
    delay(1000);
  }

}
```

Quiz

1. Wskaż zdanie nieprawdziwe
   a) Pinu przycisku monostabilnego nie trzeba konfigurować na początku programu
   b) Przycisk monostabilny po przyciśnięciu pozostaje włączony do ponownego wciśnięcia
   c) Na pinie przycisku monostabilnego w czasie, gdy jest on wyłączony, utrzymywany jest poziom niski (`LOW`)
   d) Na pinie przycisku monostabilnego w czasie, gdy jest on włączony, utrzymywany jest poziom wysoki (`HIGH`)
2. Która deklaracja stałej jest poprawna?
   a) `#define SW_CENTER 8`
   b) `#define SW_CENTER=8`
   c) `#define sw_center 8`
   d) `#define sw_center=8`
3. Który z poniższych napisów jest poprawnym warunkiem (testem) instrukcji `if` dającym informację o tym, czy przycisk lewy został wciśnięty?
   a) `if (digitalRead(SW_LEFT) = HIGH)`
   b) `if (digitalRead(SW_LEFT) == HIGH)`
   c) `if (digitalRead(SW_LEFT) = LOW)`
   d) `if (digitalRead(SW_LEFT) == LOW)`
4. Wskaż deklarację zmiennej całkowitej (typu `int`):
   a) `int liczba1;`
   b) `liczba1 int;`
   c) `int liczba 1;`
   d) `int LICZBA 1;`
5. Który z poniższych tekstów nie jest operacją na zmiennych w języku C++?
   a) `liczba=0;`
   b) `liczba=5+liczba;`
   c) `5+liczba=liczba;`
   d) `liczba=2*liczba;`

Lekcja 3

Hallo, tu Buzzer
Dźwięki w zestawie edukacyjnym TME-EDU-ARD-2

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie umiejętności programowania przycisków wbudowanych w płytkę zestawu edukacyjnego TME-EDU-ARD-2. Utrwalenie umiejętności stosowania instrukcji warunkowej jeżeli. Utrwalenie wiedzy nt. stosowania zmiennych całkowitych.

Efekty kształcenia

- Uczeń stosuje w swoich projektach przyciski monostabilne.
- Uczeń potrafi napisać program odczytujący stan przycisku monostabilnego.
- Uczeń potrafi stosować zmienne całkowite (`int`).
- Uczeń potrafi stosować instrukcję warunkową jeżeli (`if`).
- Uczeń wie, do czego stosowany jest buzzer.
- Uczeń potrafi emitować za pomocą buzzera dźwięk o określonej częstotliwości.

Wstęp

Trzecia lekcja w naszym cyklu utrwala umiejętności posługiwania się przyciskami monostabilnymi oraz diodą LED D1.

Ponadto, podczas tej lekcji poznamy buzzer i nauczymy się emitować dźwięki z układu TME--EDU-ARD-2. Ta umiejętność pozwoli nam tworzyć jeszcze ciekawsze projekty. Dzięki temu kontakt z użytkownikiem układu poza sygnałami świetlnymi na LED'ach i napisami na wyświetlaczach (wyświetlacze poznamy na kolejnych lekcjach) będzie mógł być urozmaicony dźwiękami.

Dźwięki mogą w naszych projektach towarzyszyć np. sygnalizacji poprawnego zakończenia programu, udanego ruchu/działania w grze, którą napiszemy, czy chociażby użyciu przycisków monostabilnych.

W tej lekcji wszystkie powyższe podzespoły połączymy w programie, którym utrwalimy umiejętność stosowania instrukcji warunkowej, stosowania zmiennej i wykonywania prostych operacji arytmetycznych na zmiennych.

Przebieg zajęć

1. Przedstawienie celów lekcji przez nauczyciela.

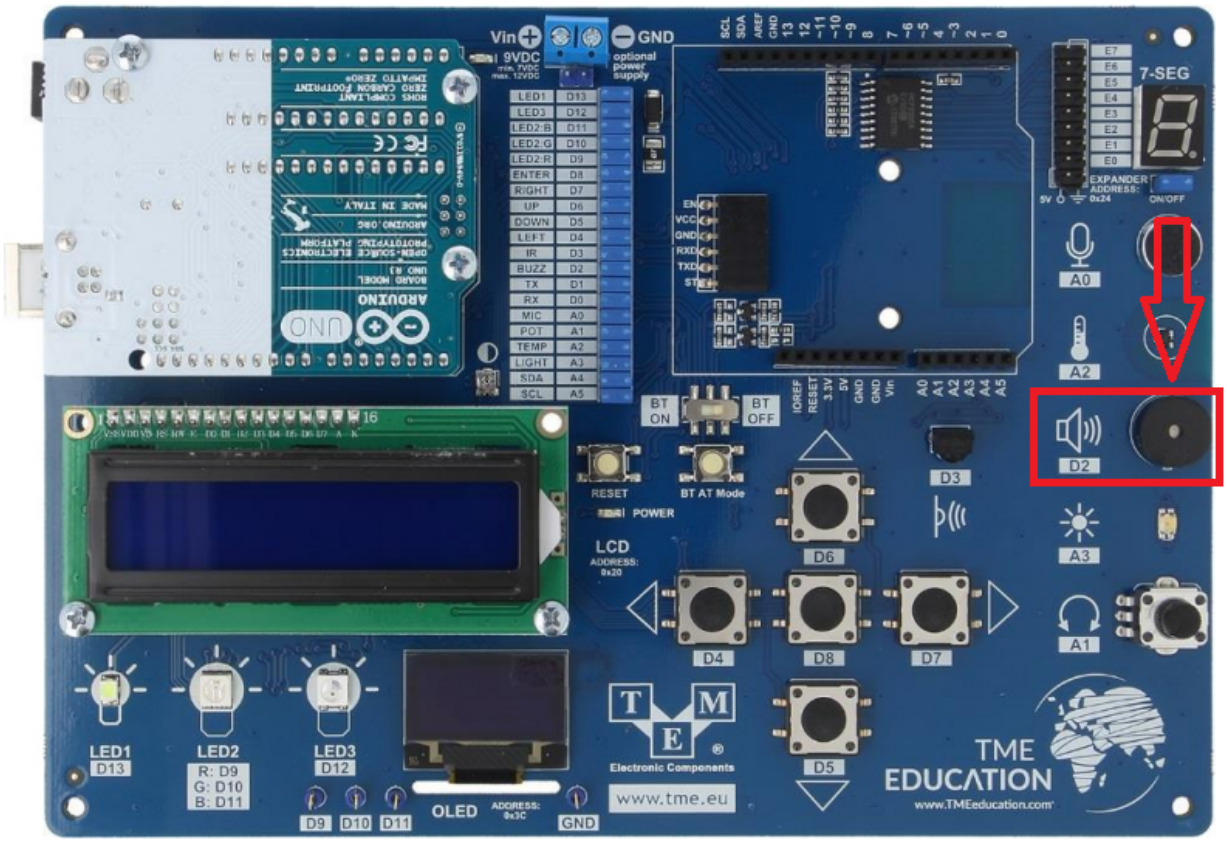
2. Powtórzenie z uczniami, jak działały następujące funkcje i konstrukcje języka:

1. `void setup()`
2. `void loop()`
3. `pinMode(13, OUTPUT);`
4. `digitalWrite(13, HIGH);`
5. `digitalWrite(13, HIGH);`
6. instrukcja warunkowa `if`
7. deklaracja zmiennej typu integer `int`

3. Powtórzenie z uczniami, w jaki sposób tworzymy nowy projekt.

4. Powtórzenie z uczniami, co musimy sprawdzić i wykonać, by wgrać nasz program do płytki Arduino UNO.

5.



Na powyższej ilustracji zaznaczony został element buzzera. Buzzer· to niewielkiej wielkości podzespół, który zachowuje się jak brzęczyk. Wbudowany element jest wyposażony w generator elektromagnetyczny , dlatego emituje jednostajny dźwięk, o stałej częstotliwości.

Jeśli chcemy, by nasz program wyemitował dźwięk z buzzera, wystarczy, że na jego wyprowadzeniu (D2) ustawimy stan wysoki (HIGH). Czyli możemy powiedzieć, że obsługa buzzera jest zbliżona do obsługi diody świecącej LED.

Jednak nie w każdym projekcie i nie dla każdej opcji w naszych programach potrzebujemy takiego samego tonu. Co zatem możemy zrobić z jednostajnie brzęczącym buzzerem? Wystarczy

go cyklicznie włączać i wyłączać, czyli ustawiać na nim stan wysoki i niski. Między poszczególnymi stanami buzzera ustawiamy opóźnienie w programie (funkcja `delay()`). Sprawi to, że buzzer będzie emitował ton o innej częstotliwości. Możemy też użyć dedykowanej do zmiany tonu funkcji języka C++, `tone()`. O jej wykorzystaniu opowiemy przy realizacji ćwiczenia.

6. Ćwiczenie 1. Do realizacji przez nauczyciela wraz z uczniami.

Napisz program, który włączy buzzer na jedną sekundę. Po tym czasie buzzer zostanie wyłączony i włączy się ponownie po kolejnej sekundzie.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
// definiujemy stałą BUZZER,
// która reprezentować będzie wskazanie pinu, na którym jest umieszczony podzespół
#define BUZZER 2
// wykonanie funkcji konfigurującej układ i jego wyprowadzenia
// przypominamy: funkcja ta uruchomi się tylko raz
void setup(){
  // ustawienie pinu buzzera jako wyjście
  pinMode(BUZZER, OUTPUT);
}

// funkcja loop() zawiera instrukcje powtarzające się w pętli nieskończonej,
// czyli wykonuje się aż do zaniku napięcia
void loop(){
  // ustawienie na wyprowdzeniu buzzera stanu wysokiego (włączenie buzzera)
  digitalWrite(BUZZER, HIGH);
  // wstrzymanie programu na 1000 ms czyli 1 sekundę
  delay(1000);
  // ustawienie na wyprowdzeniu buzzera stanu niskiego (wyłączenie buzzera)
  digitalWrite(BUZZER, LOW);
  delay(1000);
}
```

W programie zdefiniowano stałą `BUZZER`, która ułatwia nam odwołanie się do pinu, na którym w układzie Arduino UNO podłączony jest nasz brzęczyk. Podobnie jak dla diody LED, konieczne jest skonfigurowanie wyprowadzenia pinu buzzera tak, by stanowił wyjście układu. To zadanie zostało wykonane w funkcji `void setup()`.

Każde wykonanie funkcji `void loop()` zaczyna się ustawieniem stanu wysokiego `HIGH`, czyli włączeniem układu brzęczyka. Następnie, po jednosekundowej emisji spowodowanej wstrzymaniem układu przez funkcję `delay(1000)`, następuje wyłączenie generatora dźwięku. Kolejna sekunda wstrzymania programu kończy pojedynczy cykl wykonania funkcji `void loop()`. Funkcja `void loop()`, jak to było opisywane w poprzednich lekcjach, powtarza się cyklicznie, aż do wyłączenia zasilania.

7. Ćwiczenie 2. Do realizacji samodzielnie przez uczniów.

Napisz program, który włączy buzzer w chwili, kiedy zostanie naciśnięty przycisk monostabilny D8 (środkowy). W momencie zwolnienia przycisku buzzer powinien się wyłączyć.

Kod źródłowy rozwiązania ćwiczenia:

```
// definiujemy stałą BUZZER,
// która reprezentować będzie wskazanie pinu, na którym jest umieszczony podzespół
#define BUZZER 2
#define SW_CENTER 8

// wykonanie funkcji konfigurującej układ i jego wyprowadzenia
// przypominamy: funkcja ta uruchomi się tylko raz
void setup(){
  // ustawienie pinu buzzera jako wyjście
  pinMode(BUZZER, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// funkcja loop() zawiera instrukcje powtarzające się w pętli nieskończonej,
// czyli wykonuje się aż do zaniku napięcia
void loop(){
  // sprawdzenie czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH)
  {
    // ustawienie na wyprowdzeniu buzzera stanu wysokiego
    // (włączenie buzzera)
    digitalWrite(BUZZER, HIGH);
  }
  else
  {
    // ustawienie na wyprowdzeniu buzzera stanu niskiego
    // (wyłączenie buzzera)
    digitalWrite(BUZZER, LOW);
  }
}
```

W programie dodatkowo zdefiniowano stałą `SW_CENTER` przechowującą numer pinu środkowego przycisku monostabilnego. Przycisk ten trzeba było również skonfigurować w funkcji `void setup()`, tak by pin, na którym podłączono przycisk, ustawiony był jako wejściowy.

W funkcji `void loop()` zastosowano znaną z lekcji 2 instrukcję warunkową. Dzięki niej sprawdzamy stan napięcia na pinie przycisku. Jeśli jest wysoki (`HIGH`), oznacza to, że przycisk jest wciśnięty. Oznacza to dla nas, że musimy wzbudzić buzzer (`digitalWrite(BUZZER, HIGH)`). W przeciwnym wypadku, jeśli przycisk jest wyłączony, ustawiamy buzzer w stanie niskim (wyłączonym – `LOW`).

8. Ćwiczenie 3. Do realizacji z nauczycielem.

Napisz program, który w chwili wciśnięcia przycisku monostabilnego D8 (środkowy) wyemi-

tuje gamę C-dur: do-re-mi-fa-sol-la-si-do (C-D-E-F-G-A-H-C)[1]. Kolejne dźwięki powinny być emitowane co jedną sekundę.

Kod źródłowy rozwiązania ćwiczenia:

```
// definiujemy stałą BUZZER,
// która reprezentować będzie wskazanie pinu, na którym jest umieszczony podzespół
#define BUZZER 2
#define SW_CENTER 8

// Poniżej definicja częstotliwości dźwięków C, D, E, F, G, A, H, C dla przykładowej oktawy
#define C 2093
#define D 2349
#define E 2637
#define F 2794
#define G 3136
#define A 3520
#define H 3951
// wykonanie funkcji konfigurującej układ i jego wyprowadzenia
// przypominamy: funkcja ta uruchomi się tylko raz
void setup(){
  // ustawienie pinu buzzera jako wyjście
  pinMode(BUZZER, OUTPUT);
  // ustawienie pinu przycisku jako wejściowy
  pinMode(SW_CENTER, INPUT);
}

// funkcja loop() zawiera instrukcje powtarzające się w pętli nieskończonej,
// czyli wykonuje się aż do zaniku napięcia
void loop(){
  // sprawdzenie czy przycisk jest wciśnięty
  if (digitalRead(SW_CENTER) == HIGH)
  {
    // ustawienie na wyprowdzeniu buzzera stanu wysokiego
    // (włączenie buzzera z częstotliwością dźwięku C)
    tone(BUZZER, C);
    // wstrzymanie programu na 1 sekundę przed wyknaniem kolejnego dźwięku
    delay(1000);
    // (włączenie buzzera z częstotliwością kolejnych dźwięków)
    tone(BUZZER, D);
    delay(1000);
    tone(BUZZER, E);
    delay(1000);
    tone(BUZZER, F);
    delay(1000);
```

[1] Przykładowa oktawa jest w muzyce zwana czterokreślną, a jej dźwięki oznacza się: c^4, d^4, e^4, f^4, g^4, a^4, h^4. W niniejszych Lekcjach oznaczamy je wielkimi literami zgodnie z konwencją języka C++.

```
      tone(BUZZER, G);
      delay(1000);
      tone(BUZZER, A);
      delay(1000);
      tone(BUZZER, H);
      delay(1000);
      tone(BUZZER, C);
      delay(1000);
      // wyłączenie dźwięków tonów emitowanych z buzzera
      noTone(BUZZER);
    }
    else
    {
      // ustawienie na wyprowdzeniu buzzera stanu niskiego
      // (wyłączenie buzzera)
      noTone(BUZZER);
    }
  }
```

Na początku programu zdefiniowano stałe `C`, `D`, `E`, `F`, `G`, `A` oraz `H`, które reprezentują wysokości poszczególnych dźwięków gamy w przykładowej oktawie (częstotliwości w Hz, a=440, 12EDO).

Funkcja `void setup()`, podobnie jak w ćwiczeniu 2, zawiera ustawienia pinów buzzera i przycisku.

W funkcji `void loop()` w chwili naciśnięcia przycisku rozpoczyna się emitowanie gamy. Emisja dźwięków realizowana jest za pomocą funkcji `tone()`. Funkcja `tone()` może przyjmować dwa lub trzy parametry: Pin w układzie Arduino, na którym jest podłączony buzzer, częstotliwość emisji dźwięku, długość (czas trwania) dźwięku.

W przypadku przykładowego programu wykorzystano funkcję `tone()` z dwoma parametrami. Przykładowo `tone(BUZZER, C)`, gdzie w parametrze wskazano pin do emisji dźwięku i częstotliwość tonu C zapisaną w stałej zdefiniowanej na początku programu.

Na koniec bloku instrukcji emitującej dźwięki musimy zakończyć emisję ostatniego tonu. Służy do tego funkcja `noTone(BUZZER)`. Brak tej funkcji skutkować będzie emitowaniem dźwięku C, aż do momentu ponownego przyciśnięcia przycisku D8.

9. Ćwiczenie 4. Do realizacji samodzielnej przez uczniów.

Napisz program, który w chwili wciśnięcia przycisku monostabilnego D4 (lewo) wyemituje gamę C-dur w kolejności do-si-la-sol-fa-mi-re-do (C-H-A-G-F-E-D-C). Kiedy użytkownik wciśnie przycisk D7 (prawo), wyemitowana zostanie gama w kolejności do-re-mi-fa-sol-la-si-do (C-D-E-F-G-A-H-C). Kolejne dźwięki powinny być emitowane co pół sekundy.

Kod źródłowy rozwiązania ćwiczenia:

```
#define BUZZER 2
#define SW_RIGHT 7
#define SW_LEFT 4
```

```
#define C 2093
#define D 2349
#define E 2637
#define F 2794
#define G 3136
#define A 3520
#define H 3951

void setup(){
  pinMode(BUZZER, OUTPUT);
  // ustawienie pinu przycisków jako wejściowe
  pinMode(SW_RIGHT, INPUT);
  pinMode(SW_LEFT, INPUT);
}

void loop(){
  if (digitalRead(SW_RIGHT) == HIGH)
  {
    // ustawienie na wyprowdzeniu buzzera stanu wysokiego
    // (włączenie buzzera z częstotliwością dźwięku C)
    tone(BUZZER, C);
    // wstrzymanie programu na pół sekundy przed wyknaniem kolejnego dźwięku
    delay(500);
    // (włączenie buzzera z częstotliwością kolejnych dźwięków)
    tone(BUZZER, D);
    delay(500);
    tone(BUZZER, E);
    delay(500);
    tone(BUZZER, F);
    delay(500);
    tone(BUZZER, G);
    delay(500);
    tone(BUZZER, A);
    delay(500);
    tone(BUZZER, H);
    delay(500);
    tone(BUZZER, C);
    delay(500);
    // wyłączenie dźwięków tonów emitowanych z buzzera
    noTone(BUZZER);
  }
  if (digitalRead(SW_LEFT) == HIGH)
  {
    tone(BUZZER, C);
    delay(500);
    tone(BUZZER, H);
```

```
      delay(500);
      tone(BUZZER, A);
      delay(500);
      tone(BUZZER, G);
      delay(500);
      tone(BUZZER, F);
      delay(500);
      tone(BUZZER, E);
      delay(500);
      tone(BUZZER, D);
      delay(500);
      tone(BUZZER, C);
      delay(500);
    noTone(BUZZER);
  }
}
```

W programie zdefiniowano stałe na oznaczenie przycisków prawo/lewo. Dodano drugą instrukcję warunkową, w której obsługiwana jest emisja dźwięków gamy w odwrotnej kolejności.

10. Ćwiczenie 5. **Dla zaawansowanych**. Zamek szyfrowy z alarmem. Do realizacji samodzielnej przez uczniów.

Napisz program, który w chwili wciśnięcia sekwencji przycisków: lewy, środkowy i prawy (poprawny kod) wyemituje 3 takie same dźwięki w odstępach 200 ms.

W przypadku błędnego wpisania kodu (innej sekwencji przycisków) program wyemituje wyemituje zmiennie co 500 ms dwa tony, częstotliwościach 2000 Hz i 2828 Hz[2]

Dioda LED D1 powinna zostać włączona na początku programu i świecić światłem ciągłym w czasie, kiedy program oczekuje wpisania kodu.

Kod źródłowy rozwiązania ćwiczenia:

```
// definiujemy stałą BUZZER,
// która reprezentować będzie wskazanie pinu, na którym jest umieszczony podzespół
#define BUZZER 2
// definiujemy pin diody LED
#define LED 13
// definiujemy piny przycisków
#define SW_RIGHT 7
#define SW_CENTER 8
#define SW_LEFT 4

// Poniżej definicja wysokości dźwięków dla sygnału otwarcia zamka, czyli kodu poprawnego,
#define TONE_OPEN 1000
// ...i dwóch tonów alarmu
#define TONE_ALARM_1 2000
```

[2] tony mają tu sygnalizować, że coś poszło nie tak, dlatego proponujemy interwał odbierany jako szczególnie nieprzyjemny, zwany „kwintą diabelską” lub trytonem.

```
#define TONE_ALARM_2 2828

// deklaracja zmiennej pomocniczej, w której będziemy zapamiętywać wpisany kod
int klucz=0;

void setup(){
  // ustawienie pinu buzzera jako wyjście
  pinMode(BUZZER, OUTPUT);
  // ustawienie pinu LED'a jako wyjście
  pinMode(LED, OUTPUT);
  // ustawienie pinu przycisków jako wejście
  pinMode(SW_RIGHT, INPUT);
  pinMode(SW_CENTER, INPUT);
  pinMode(SW_LEFT, INPUT);
  // włączenie diody LED
  digitalWrite(LED, HIGH);
}

/*
*  W funkcji loop() sprawdzamy, czy wciśnięty jest któryś z przycisków. Jeśli tak, to
*  w zależności od tego, który, dodajemy do zmiennej klucz przypisaną do niego cyfrę
*  kodu, tj.:
*  • dla przycisku SW_LEFT przypisujemy 1.
*  • dla przycisku SW_CENTER przypisujemy 2.
*  • dla przycisku SW_RIGHT przypisujemy 3.
*  Operacja arytmetyczna, jaka odpowiada dopisaniu kolejnej cyfry (dziesiętnej) na końcu liczby
*  (w reprezentacji dziesiętnej), to przemnożenie tej liczby przez 10 i dodanie tej nowej cyfry.
*  Tak właśnie postąpimy ze zmienną klucz i przekazaną w postaci jednocyfrowej liczby
*  informacją o ostatnio wciśniętym przycisku.
*/
void loop(){
    // sprawdzamy, czy w zmiennej klucz są już trzy cyfry – czy wprowadzono już cały klucz
  if(klucz>111)
  {
    // sprawdzamy, czy w zmiennej klucz znajduje się poprawny kod
    if(klucz==123)
    {
      tone(BUZZER, TONE_OPEN);
      delay(200);
      noTone(BUZZER);
      delay(200);
      tone(BUZZER, TONE_OPEN);
      delay(200);
      noTone(BUZZER);
      delay(200);
      tone(BUZZER, TONE_OPEN);
```

```
      delay(200);
      noTone(BUZZER);
      delay(200);
      digitalWrite(LED, LOW);
      klucz=0;
    }
    else // liczba w zmiennej klucz nie jest poprawnym kodem
    {
      tone(BUZZER, TONE_ALARM_1);
      digitalWrite(LED, HIGH);
      delay(500);
      tone(BUZZER, TONE_ALARM_2);
      digitalWrite(LED, LOW);
      delay(500);
    }
  }
  else
  {
    digitalWrite(LED, HIGH);
  }
  if (digitalRead(SW_LEFT) == HIGH)
  {
    klucz=klucz*10+1;
    delay(500);
  }
  if (digitalRead(SW_CENTER) == HIGH)
  {
    klucz=klucz*10+2;
    delay(500);
  }
  if (digitalRead(SW_RIGHT) == HIGH)
  {
    klucz=klucz*10+3;
    delay(500);
  }
}
```

11. Przeprowadzenie quizu podsumowującego lekcję.

1. Buzzer to:
   a) przycisk
   b) brzęczyk
   c) dioda
   d) wyświetlacz
2. Podstawowe instrukcje sterujące buzzerem są podobne do instrukcji:

a) dczytu danych z przycisków
b) włączania diody LED
c) deklaracji zmiennej

3. Instrukcja `digitalWrite(BUZZER, HIGH)` spowoduje:
   a) wyłączenie dźwięku buzzera
   b) włączenie jednostajnego dźwięku ciągłego
   c) włączenie zmiennotonowego dźwięku buzzera
   d) włączenie jednego tonu na jedną sekundę
4. Wskaż poprawne użycie funkcji `tone()`
   a) `tone(500, BUZZER, TONE_ALARM_1);`
   b) `tone(TONE_ALARM_1, BUZZER);`
   c) `tone(BUZZER, TONE_ALARM_1);`

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który włączy buzzer na jedną sekundę. Po tym czasie buzzer zostanie wyłączony i włączy się ponownie po kolejnej sekundzie.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
// definiujemy stałą BUZZER,
// która reprezentować będzie wskazanie pinu, na którym jest umieszczony podzespół
#define BUZZER 2
// wykonanie funkcji konfigurującej układ i jego wyprowadzenia
// przypominamy: funkcja ta uruchomi się tylko raz
void setup(){
  // ustawienie pinu buzzera jako wyjście
  pinMode(BUZZER, OUTPUT);
}

// funkcja loop() zawiera instrukcje powtarzające się w pętli nieskończonej,
// czyli wykonuje się aż do zaniku napięcia
void loop(){
  // ustawienie na wyprowdzeniu buzzera stanu wysokiego (włączenie buzzera)
  digitalWrite(BUZZER, HIGH);
  // wstrzymanie programu na 1000 ms czyli 1 sekundę
  delay(1000);
  // ustawienie na wyprowdzeniu buzzera stanu niskiego (wyłączenie buzzera)
  digitalWrite(BUZZER, LOW);
  delay(1000);
}
```

Ćwiczenie 2

Napisz program, który włączy buzzer w chwili, kiedy zostanie naciśnięty przycisk monostabilny D8 (środkowy). W momencie zwolnienia przycisku buzzer powinien się wyłączyć.

Ćwiczenie 3

Napisz program, który w chwili wciśnięcia przycisku monostabilnego D8 (środkowy) wyemituje gamę C-dur: do-re-mi-fa-sol-la-si-do (C-D-E-F-G-A-H-C)[1]. Kolejne dźwięki powinny być emitowane co jedną sekundę.

Ćwiczenie 4

Napisz program, który w chwili wciśnięcia przycisku monostabilnego D4 (lewo) wyemituje gamę C-dur w kolejności do-si-la-sol-fa-mi-re-do (C-H-A-G-F-E-D-C). Kiedy użytkownik wciśnie przycisk D7 (prawo), wyemitowana zostanie gama w kolejności do-re-mi-fa-sol-la-si-do (C-D-E-F-G-A-H-C). Kolejne dźwięki powinny być emitowane co pół sekundy.

Ćwiczenie 5

Dla zaawansowanych. Zamek szyfrowy z alarmem.

Napisz program, który w chwili wciśnięcia sekwencji przycisków: lewy, środkowy i prawy (poprawny kod) wyemituje 3 takie same dźwięki w odstępach 200 ms.

W przypadku błędnego wpisania kodu (innej sekwencji przycisków) program wyemituje wyemituje zmiennie co 500 ms dwa tony, częstotliwościach 2000 Hz i 2828 Hz[2]

Dioda LED D1 powinna zostać włączona na początku programu i świecić światłem ciągłym w czasie, kiedy program oczekuje wpisania kodu.

[1] Przykładowa oktawa jest w muzyce zwana czterokreślną, a jej dźwięki oznacza się: c^4, d^4, e^4, f^4, g^4, a^4, h^4. W niniejszych Lekcjach oznaczamy je wielkimi literami zgodnie z konwencją języka C++.

[2] tony mają tu sygnalizować, że coś poszło nie tak, dlatego proponujemy interwał odbierany jako szczególnie nieprzyjemny, zwany „kwintą diabelską” lub trytonem.

Quiz

1. Buzzer to:
   a) przycisk
   b) brzęczyk
   c) dioda
   d) wyświetlacz
2. Podstawowe instrukcje sterujące buzzerem są podobne do instrukcji:
   a) dczytu danych z przycisków
   b) włączania diody LED
   c) deklaracji zmiennej
3. Instrukcja `digitalWrite(BUZZER, HIGH)` spowoduje:
   a) wyłączenie dźwięku buzzera
   b) włączenie jednostajnego dźwięku ciągłego
   c) włączenie zmiennotonowego dźwięku buzzera
   d) włączenie jednego tonu na jedną sekundę
4. Wskaż poprawne użycie funkcji `tone()`
   a) `tone(500, BUZZER, TONE_ALARM_1);`
   b) `tone(TONE_ALARM_1, BUZZER);`
   c) `tone(BUZZER, TONE_ALARM_1);`

Lekcja 4

Transmisja szeregowa. Przesyłamy dane z TME-EDU-ARD-2 do komputera.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych całkowitych. Utrwalenie umiejętności stosowania przełączników monostabilnych, buzzera i diody LED D1. Nabycie umiejętności stosowania interfejsu UART do komunikacji układu Arduino UNO z komputerem.

Efekty kształcenia

- Uczeń stosuje w swoich projektach przyciski monostabilne.
- Uczeń potrafi stosować zmienne całkowite (`int`).
- Uczeń potrafi stosować instrukcję warunkową jeżeli (`if`).
- Uczeń wie na czym polega transmisja szeregowa UART
- Uczeń potrafi przesyłać informację z i do komputera du układu Arduino.

Wstęp

W tej lekcji dowiesz się do na czym polega transmisja szeregowa. Lekcja 4 dla Ciebie i Twoich uczniów będzie okazją do nawiązania interakcji z użytkownikiem układu. W trakcie działania naszego programu płytka Arduino UNO będzie przesyłała wyniki obliczeń do komputera. My jako użytkownicy układu będziemy mogli wysyłać do niej dane w celu przetworzenia. Ta komunikacja będzie się odbywać przez połączenie szeregowe UART.

UART jest to transmisja przewodowa, a odbywa się ona przez ten sam przewód USB A-B do którego podłączony jest zestaw edukacyjny TME-EDU-ARD-2. Po stronie komputera nie potrzebujemy pisać żadnego dodatkowego programu, ani też ściągać dodatkowego oprogramowania. Za całą komunikację będzie odpowiedzialne środowisko w którym już programujemy, czyli Arduino IDE.
Jedyne co potrzebujemy to poprawnie napisany program na płytkę Arduino UNO i poprawne skonfigurowanie połączenia.

Przebieg zajęć

1. Przedstawienie celów lekcji.
2. Powtórzenie wiadomości z poprzednich lekcji,

1. Do czego służy funkcja `loop()` i `setup()`
2. Jak wgrywać program do układu Arduino
3. Za pomocą jakich podzespołów prowadzona była dotychczas komunikacja z użytkownikiem? Odp. Przyciski monostabilne, dioda LED D1, buzzer.
4. Jak programować ww. podzespoły?

3. Omówienie przez nauczyciela, czym jest UART.

UART jest rodzajem transmisji, który polega na szeregowym wysyłaniu sygnałów zero/jedynkowych. W tej transmisji uczestniczą dwie żyły. Jedna z nich odpowiedzialna jest za wysyłanie danych z układu lub komputera (oznaczana Tx). Druga zaś odpowiedzialna jest za odbieranie danych z komputera lub z układu Arduino(oznaczana Rx). Każdą z tych żył płyną sygnały 1 bitowe (zero -0V lub jeden – 5V). Sygnały wypływają z miejsca źródłowego z konkretną częstotliwością.

Bardzo ważne jest, aby urządzenia, które komunikują się w tym standardzie nadawały i odbierały sygnały z tą samą częstotliwością. Czyli, aby urządzenie odbiorcze wiedziało z jaką częstotliwością ma sprawdzać kolejne napływające sygnały na linii odbiorczej. Z kolei urządzenie nadawcze powinno z taką częstotliwością wysyłać kolejne bity.

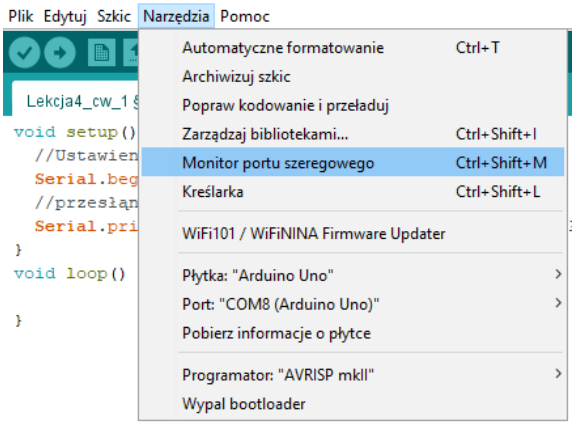
Częstotliwość transmisji będziemy wyrażać w bitach na sekundę, a nazywać ją będziemy baud-rate.

W naszych ćwiczeniach ustalmy prędkość transmisji na 9600 bitów na sekundę.

Do transmisji UART nie potrzebujemy, żadnych nowych kabli i połączeń. Wykorzystamy wpięty już do programowania przewód USB.

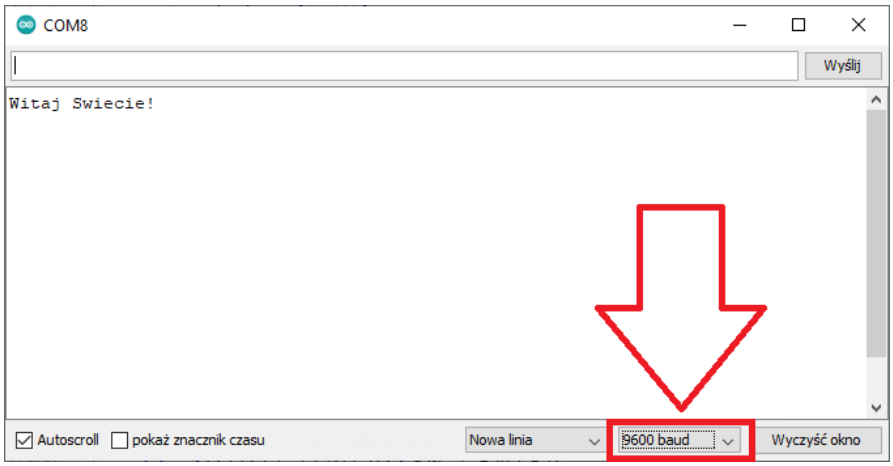
4. Konfiguracja „Monitora portu szeregowego” w środowisku Arduino IDE.

Jak wspominaliśmy, do poprawnej komunikacji konieczne jest ustawienie wspólnej częstotliwości dla układu Arduino UNO i naszego komputera. Przyjmijmy do celów realizacji tej lekcji, że transmisja odbywa się z częstotliwością 9600 bitów na sekundę.



„Monitor portu szeregowego” znajdziemy w środowisku Arduino IDE w menu Narzędzia.

Po wywołaniu tej opcji pokaże nam się okno do przeprowadzanie monitorowania i wysyłania sygnałów szeregowych UART.



W prawym dolnym rogu tego okna zmieńmy częstotliwość transmisji na 9600.

Okno możemy zostawić otwarte.

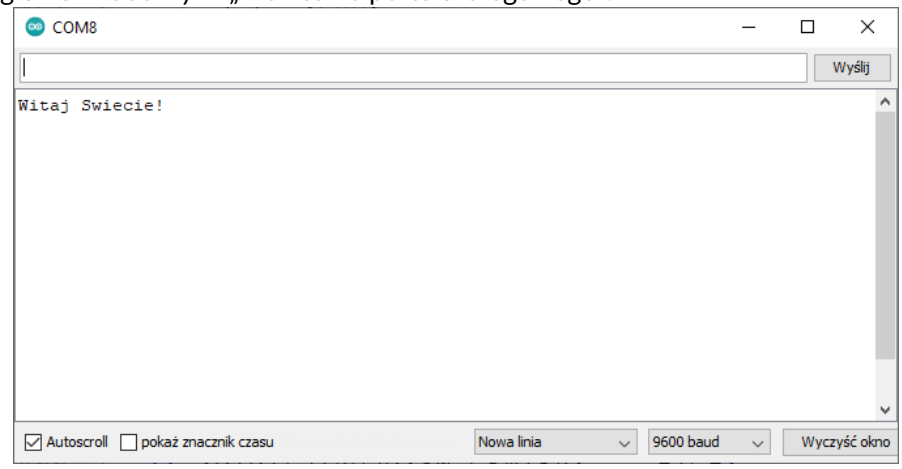
5. Ćwiczenie 1. Do realizacji wspólnie z nauczycielem.

Napisz program, który wyśle z układu Arduino UNO do komputera komunikat „Witaj Świecie”.

Kod źródłowy rozwiązania ćwiczenia:

```
void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
  // przesłanie do komputera jednej linijki tekstu
  Serial.println("Witaj Swiecie!");
}
void loop() {
}
```

Przykładowy efekt działania programu widoczny w „Monitorze portu szeregowego”:



W programie przed rozpoczęciem transmisji musimy określić jej prędkość, dlatego w funkcji `void setup()` jako pierwsza instrukcja występuje właśnie instrukcja ustawiająca prędkość transmisji na 9600 bitów na sekundę (`Serial.begin(9600)`). Druga instrukcja to wysłanie jednowierszowego komunikatu do komputera (`Serial.println("Witaj Swiecie!")`)

UWAGA! W programie wystarczy zawrzeć jedną instrukcję ustawiającą prędkość transmisji. Kolejne linie tekstu jakie chcielibyśmy przesyłać lub odbierać nie wymagają ponownej konfiguracji prędkości.

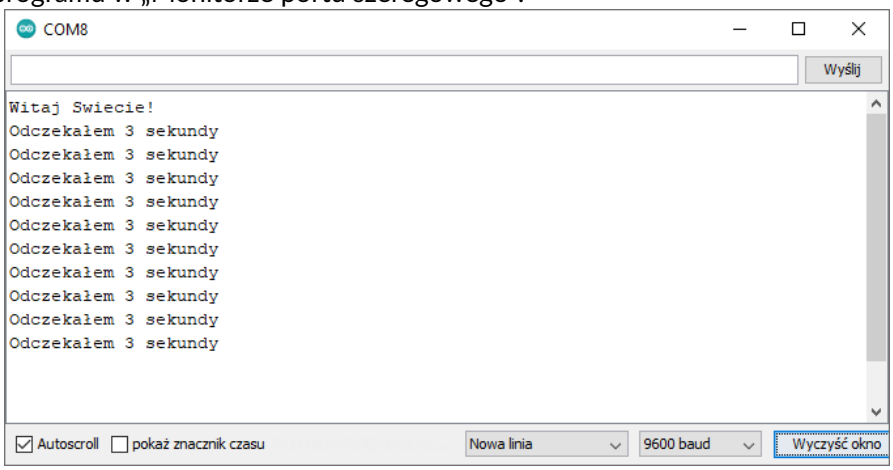
6. Ćwiczenie 2. Do samodzielnej realizacji przez uczniów

Napisz program, który wyśle z układu Arduino UNO do komputera komunikat „Witaj Swiecie". Po czym co 3 sekundy wyśle komunikat „Odczekałem 3 sekundy"

Kod źródłowy rozwiązania ćwiczenia:

```
void setup(){
  // Ustawienie częstotliwości nadawania i odbioru
  Serial.begin(9600);
  // przesłanie do komputera jednej linijki tekstu
  Serial.println("Witaj Swiecie!"); // Jednorazowe wysłanie tekstu
}
void loop() {
  // odczekanie 3 sekund
  delay(3000);
  // przesłanie do komputera jednej linijki tekstu
  Serial.println("Odczekałem 3 sekundy"); // Jednorazowe wysłanie tekstu
}
```

Przykładowe okno działającego programu w „Monitorze portu szeregowego":



7. Ćwiczenie 3. Do samodzielnej realizacji przez uczniów

Zmodyfikuj program z Ćwiczenia 2 tej lekcji tak, aby przyciskami monostabilnymi ustalać czas odstępu między kolejnymi komunikatami.

Naciśnięcie:

- Przycisk `SW_LEFT` – odstęp 1 sekunda
- Przycisk `SW_CENTER` – odstęp 2 sekundy
- Przycisk `SW_RIGHT` – odstęp 3 sekundy

Komunikat „Odczekałem 3 sek." musi być odpowiedni dla każdego z odstępów czasowych.

W chwili wykrycia wciśniętego przycisku powinien się pojawić na „Monitorze portu szeregowego" komunikat: „Zmieniono opoznienie na .....".

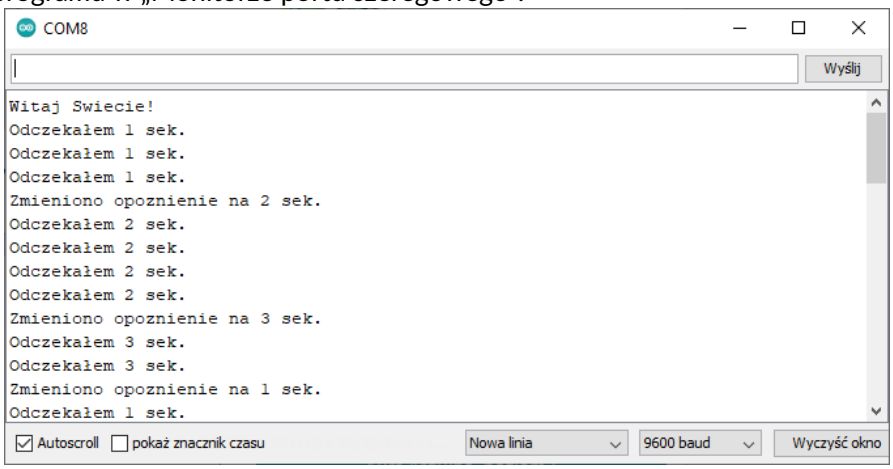
Gdy program uruchomi się, opóźnienie powinno być ustawione początkowo na 1 sekundę.

Kod źródłowy rozwiązania ćwiczenia:

```
#define SW_RIGHT 7
#define SW_CENTER 8
#define SW_LEFT 4
int predkosc=1000;
void setup(){
  // ustawienie pinu przycisków jako wejściowe
  pinMode(SW_RIGHT, INPUT);
  pinMode(SW_CENTER, INPUT);
  pinMode(SW_LEFT, INPUT);
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
  // przesłanie do komputera jednej linijki tekstu
  Serial.println("Witaj Swiecie!");
}
void loop() {
  if (digitalRead(SW_LEFT) == HIGH)
  {
    predkosc=1000;
    Serial.println("Zmieniono opoznienie na 1 sek.");
  }
  if (digitalRead(SW_CENTER) == HIGH)
  {
    predkosc=2000;
    Serial.println("Zmieniono opoznienie na 2 sek.");
  }
  if (digitalRead(SW_RIGHT) == HIGH)
  {
    predkosc=3000;
    Serial.println("Zmieniono opoznienie na 3 sek.");
  }

  // odczekanie x sekund
  delay(predkosc);
  if(predkosc==1000)
  {
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odczekałem 1 sek.");
  }

  if(predkosc==2000)
  {
```

```
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odczekałem 2 sek.");
  }

  if(predkosc==3000)
  {
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odczekałem 3 sek.");
  }
}
```

Przykładowe okno działającego programu w „Monitorze portu szeregowego”:



W programie zastosowano zmienną typu całkowitego o nazwie `predkosc`. Początkowo do zmiennej przypisano wartość 1000, co oznacza 1000 milisekund opóźnienia przed kolejnymi transmisjami wiadomości.

Instrukcje warunkowe znajdujące się w funkcji `void loop()` sprawdzają, czy nie został naciśnięty przycisk monostabilny. Gdyby wykryto wciśnięcie przycisku `SW_LEFT`, `SW_CENTER`, `SW_RIGHT`, zmieniona zostanie wartość zmiennej predkosc odpowiednio na 1000, 2000, 3000 (milisekund).

8. Omówienie przez nauczyciela typu znakowego `char` oraz łańcuchowego `string` w języku C++.

W programach w języku C++ poza pracą na liczbach możliwe jest również przetwarzanie danych znakowych. Umiejętność stosowania zmiennych znakowych przyda nam się, kiedy pracujemy z transmisją szeregową UART i chcemy odbierać w układzie Arduino UNO dane od użytkownika komputera.

Jeżeli w naszym programie będziemy chcieli pracować z pojedynczym znakiem możemy zastosować zmienną znakową char. Zmienna char podobnie jak zmienna typu integer musi być przed pierwszym użyciem zadeklarowana.

Przykładowa deklaracja zmiennej znakowej:

```
char znak;
```

Jeśli chcemy w programie przypisać wartość zmiennej znakowej musimy przypisywany znak ująć w apostrofy, np.:

```
znak='a';
```

W naszych programach częściej stosować będziemy zmienne składające się z wielu znaków, czyli zmienne łańcuchowe – typu `string`. Ich deklaracja przebiega podobnie.

```
string slowo;
```

Jednak przypisanie wartości do zmiennej łańcuchowej odbywa się przez ujęcie przypisywanego łańcucha znaków w cudzysłów (podwójny):

```
slowo="Zestaw edukacyjny";
```

9. Ćwiczenie 4

Napisz program, który będzie odbierał od użytkownika sygnały wysyłane w „Monitorze portu szeregowego". Tymi sygnałami będą cyfry 0 lub 1.

Jeśli odebranym sygnałem będzie „1" to program powinien włączyć diodę LED D1.

Jeśli odebranym sygnałem będzie „0" to program powinien wyłączyć diodę LED D1

Kod źródłowy rozwiązania ćwiczenia:

```
#define LED1 13

string sygnal="0";

void setup(){
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
}

void loop() {
  if(Serial.available() > 0)
  {
    sygnal = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
  }

  if(sygnal=="0")
  {
    digitalWrite(LED1, LOW);
  }

  else
  {
    digitalWrite(LED1, HIGH);
```

```
    }
  }
```

W programie tym w funkcji `void loop()` użyto instrukcji warunkowej `if(Serial.available() > 0)`, która sprawdza czy w buforze transmisji szeregowej znajdują się jakieś znaki, które można odebrać. Jeśli w buforze jest więcej znaków niż zero to do zmiennej string przypisujemy kolejne znaki z bufora, aż do znaku przejścia do nowego wiersza. W naszym przypadku spodziewamy się jedynie znaków „0" lub „1" dlatego w kolejnej instrukcji warunkowej sprawdzamy właśnie jaką wartość ma zmienna sygnał i odpowiednio dla „0" wyłączamy diodę LED D1, a dla „1" włączamy diodę LED D1.

10. Ćwiczenie 5

Zmodyfikuj program z ćwiczenia 4 tak by wysyłał do użytkownika po UART komunikat o błędnym poleceniu (sygnale).

Dodatkowo wprowadź do programu pojedynczy sygnał z buzzera w sytuacji poprawnego polecenia „0" lub „1". Błędne polecenie (naciśnięcie przycisku innego niż „0" lub „1") powinno wywołać podwójny sygnał buzzera.

Kod źródłowy rozwiązania ćwiczenia:

```
// definiujemy stałą buzzer
// która reprezentować będzie wskazanie pinu na którym jest umieszczony podzespół
#define BUZZER 2
#define TONE 1000
// definiujemy pin diody LED
#define LED1 13

string sygnal="0";

void setup(){
  // ustawienie pinu buzzera jako wyjście
  pinMode(BUZZER, OUTPUT);
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
}

void loop() {
  if(Serial.available() > 0)
  {
    sygnal = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
    if(sygnal=="0")
    {
      // generujemy pojedynczy sygnał z buzzera
      tone(BUZZER, TONE);
```

```
      delay(200);
      noTone(BUZZER);
    }
    else if(sygnal=="1")
    {
      // generujemy pojedynczy sygnał z buzzera
      tone(BUZZER, TONE);
      delay(200);
      noTone(BUZZER);
    }
     else
    { // generujemy podwójny sygnał z buzzera
      tone(BUZZER, TONE);
      delay(200);
      noTone(BUZZER);
      delay(200);
      tone(BUZZER, TONE);
      delay(200);
      noTone(BUZZER);
    }
  }

  if(sygnal=="0")
  {
    digitalWrite(LED1, LOW);
  }
  else if(sygnal=="1")
  {
    digitalWrite(LED1, HIGH);
  }
}
```

W programie w sytuacji, kiedy wykryjemy, że w buforze transmisji szeregowej czekają jakieś znaki (`if(Serial.available() > 0)`) odbieramy je i przypisujemy do zmiennej sygnał (`sygnal = Serial.readStringUntil('\n');`). W tym samym momencie, kiedy odbierzemy znaki sprawdzamy, czy odebrane znaki były „0" czy „1" jeśli tak to generujemy wtedy pojedynczy sygnał z buzzera. W przeciwnym wypadku z buzzerA rozlegają się dwa wygenerowane w ostępie 200 ms dźwięki.

Ponadto dla sygnału „0" wykonujemy polecenie `digitalWrite(LED1, LOW)` wyłączające diodę LED. Dla sygnału „1" wykonujemy polecenie `digitalWrite(LED1, HIGH)` włączające diodę LED.

11. Przeprowadzenie quizu podsumowującego lekcję.

1. UART to:
   a) Podzespół na płytce edukacyjnej TME-EDU-ARD-2
   b) Szeregowa transmisja danych

c) Rodzaj wyświetlacza
d) Polecenie w C++

2. Do poprawnej komunikacji szeregowej komputera z Arduino UNO nie będzie niezbędne:
   a) Ustawienie prędkości nadawania danych
   b) Ustawienie prędkości odbierania danych
   c) Kabel USB
   d) Dioda LED D1
3. Instrukcja `Serial.begin(9600);` ...
   a) Powinna być wykonana na początku działania programu
   b) Powinna być wykonana przed każdą instrukcją wysyłającą dane
   c) Zawsze w parametrze zawiera liczbę 9600
   d) Jest zbędna i można ją pominąć przy transmisji
4. Instrukcja `Serial.available()` służy do:
   a) Ustawienia prędkości transmisji szeregowej
   b) Sprawdzania, czy obsługa transmisji szeregowej jest dostępna w tej wersji układu
   c) Sprawdzenia, czy układ działa poprawnie.
   d) Sprawdzenia, czy są jakieś dane w buforze odczytu.
5. Do odbierania danych z UART użyjemy instrukcji:
   a) `Serial.readStringUntil('\n');`
   b) `Serial.available();`
   c) `digitalWrite(Serial);`
   d) `Serial.begin(9600);`

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

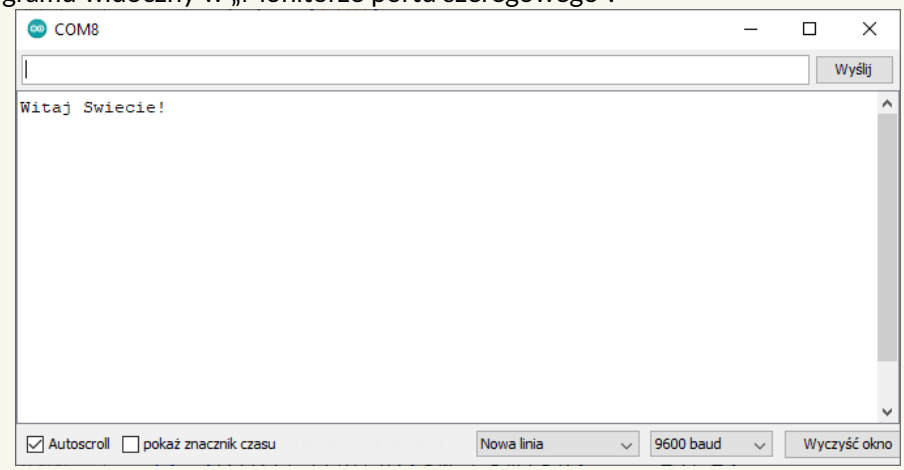
Ćwiczenie 1

Napisz program, który wyśle z układu Arduino UNO do komputera komunikat „Witaj Świecie”.

Kod źródłowy rozwiązania ćwiczenia:

```
void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
  // przesłanie do komputera jednej linijki tekstu
  Serial.println("Witaj Swiecie!");
}
void loop() {
}
```

Przykładowy efekt działania programu widoczny w „Monitorze portu szeregowego”:



Ćwiczenie 2

Napisz program, który wyśle z układu Arduino UNO do komputera komunikat „Witaj Swiecie”. Po czym co 3 sekundy wyśle komunikat „Odczekałem 3 sekundy”

Ćwiczenie 3

Zmodyfikuj program z Ćwiczenia 2 tej lekcji tak, aby przyciskami monostabilnymi ustalać czas odstępu między kolejnymi komunikatami.

Naciśnięcie:

— Przycisk `SW_LEFT` – odstęp 1 sekunda
— Przycisk `SW_CENTER` – odstęp 2 sekundy
— Przycisk `SW_RIGHT` – odstęp 3 sekundy

Komunikat „Odczekałem 3 sek.” musi być odpowiedni dla każdego z odstępów czasowych.

W chwili wykrycia wciśniętego przycisku powinien się pojawić na „Monitorze portu szeregowego” komunikat: „Zmieniono opoznienie na .....”.

Gdy program uruchomi się, opóźnienie powinno być ustawione początkowo na 1 sekundę.

Ćwiczenie 4

Napisz program, który będzie odbierał od użytkownika sygnały wysyłane w „Monitorze portu szeregowego”. Tymi sygnałami będą cyfry 0 lub 1.

Jeśli odebranym sygnałem będzie „1” to program powinien włączyć diodę LED D1.

Jeśli odebranym sygnałem będzie „0” to program powinien wyłączyć diodę LED D1

Kod źródłowy rozwiązania ćwiczenia:

```
#define LED1 13

string sygnal="0";

void setup(){
  // ustawienie pinu diody LED jako wyjściowy
  pinMode(LED1, OUTPUT);
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
}

void loop() {
  if(Serial.available() > 0)
  {
    sygnal = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
  }

  if(sygnal=="0")
  {
    digitalWrite(LED1, LOW);
  }

  else
  {
    digitalWrite(LED1, HIGH);
  }

}
```

Ćwiczenie 5

Zmodyfikuj program z ćwiczenia 4 tak by wysyłał do użytkownika po UART komunikat o błędnym poleceniu (sygnale).

Dodatkowo wprowadź do programu pojedynczy sygnał z buzzera w sytuacji poprawnego polecenia „0” lub „1”. Błędne polecenie (naciśnięcie przycisku innego niż „0” lub „1”) powinno wywołać podwójny sygnał buzzera.

Quiz

1. UART to:
   a) Podzespół na płytce edukacyjnej TME-EDU-ARD-2
   b) Szeregowa transmisja danych
   c) Rodzaj wyświetlacza
   d) Polecenie w C++
2. Do poprawnej komunikacji szeregowej komputera z Arduino UNO nie będzie niezbędne:
   a) Ustawienie prędkości nadawania danych
   b) Ustawienie prędkości odbierania danych
   c) Kabel USB
   d) Dioda LED D1
3. Instrukcja `Serial.begin(9600);` ...
   a) Powinna być wykonana na początku działania programu
   b) Powinna być wykonana przed każdą instrukcją wysyłającą dane
   c) Zawsze w parametrze zawiera liczbę 9600
   d) Jest zbędna i można ją pominąć przy transmisji
4. Instrukcja `Serial.available()` służy do:
   a) Ustawienia prędkości transmisji szeregowej
   b) Sprawdzania, czy obsługa transmisji szeregowej jest dostępna w tej wersji układu
   c) Sprawdzenia, czy układ działa poprawnie.
   d) Sprawdzenia, czy są jakieś dane w buforze odczytu.
5. Do odbierania danych z UART użyjemy instrukcji:
   a) `Serial.readStringUntil('\n');`
   b) `Serial.available();`
   c) `digitalWrite(Serial);`
   d) `Serial.begin(9600);`

Lekcja 5

Prosty kalkulator i liczenie średniej arytmetycznej w zestawie edukacyjnym TME-EDU-ARD-2.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych całkowitych, instrukcji warunkowych w języku C++. Zdobycie umiejętności programowania wyświetlacza LCD. Nabycie wiedzy i ukształtowanie umiejętności stosowania zmiennych rzeczywistych w języku C++. Zdobycie wiedzy nt. dołączania bibliotek w języku C++. Ukształtowanie umiejętności instalacji dodatkowych bibliotek w środowisku Arduino IDE.

Efekty kształcenia

- Uczeń stosuje przyciski monostabilne w swoich programach.
- Uczeń stosuje transmisję UART do komunikacji układu Arduino z komputerem.
- Uczeń zna pojęcie zmiennej rzeczywistej.
- Uczeń potrafi stosować w zmienne rzeczywiste (`float`).
- Uczeń wie, jak dołączyć do projektu Arduino bibliotekę obsługującą urządzenia peryferyjne.

Wstęp

W trakcie lekcji utrwalimy umiejętności posługiwania się połączeniem UART i dotychczas poznanymi peryferiami zestawu edukacyjnego TME-EDU-ARD-2. Wykonywane ćwiczenia przedstawią kilka sposobów na realizację prostego kalkulatora z użyciem układu Arduino i naszego zestawu edukacyjnego. W trakcie pracy nad kalkulatorem będziemy odbierać dane do operacji za pomocą połączenia UART, a w kolejnych ćwiczeniach za pomocą przycisków monostabilnych. Wyniki zaś zobaczymy w pierwszych ćwiczeniach w „Monitorze portu szeregowego” poznanym na ostatniej lekcji. Jednak najciekawszy sposób prezentacji dający ogromne możliwości poznamy w kolejnych ćwiczeniach tej lekcji. Będziemy tam wyświetlać wyniki i nasze działania za pomocą wyświetlacza LCD.

Ponadto poza znaną nam już z poprzednich lekcji zmienną całkowitą integer, nauczymy się pracować na zmiennych rzeczywistych `float` tj. takich dla których poza częścią całkowitą będziemy mogli zapisać i obsługiwać również część ułamkową.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

1. Do czego służy funkcja loop() i setup()
2. Jak wgrywać program do układu Arduino
3. Za pomocą jakich podzespołów prowadzona była dotychczas komunikacja z użytkownikiem? Odp. Przyciski monostabilne, dioda LED D1, buzzer.
4. Jak programować poznane na poprzednich lekcjach podzespoły, tj. Przyciski monostabilne, dioda LED D1, buzzer?
5. Powtórzenie przez nauczyciela, czym jest UART. Jak należy konfigurować UART w programie Arduino i na komputerze w „Monitorze portu szeregowego"

3. Omówienie przez nauczyciela instrukcji pobierającej przez układ Arduino zmiennej całkowitej z komputera.

Jak pamiętamy z lekcji 4 do interakcji z użytkownikiem naszego zestawu możemy posłużyć się nie tylko peryferiami zestawu edukacyjnego, ale również połączeniem UART z naszym komputerem.

Aby nawiązać połączenie UART musimy ustalić wspólną prędkość transmisji dla odbioru i wysyłania danych. Najwygodniej wykonać taką konfigurację w programie zaraz po starcie układu czyli w funkcji `void setup()`, gdzie użyjemy instrukcji `Serial.begin(9600)`. Instrukcja ta ustali prędkość transmisji na 9600 bitów na sekundę. Taką samą prędkość należy ustawić w „Monitorze portu szeregowego". Dostępnym w menu Narzędzia środowiska Arduino IDE.

Odbiór danych w transmisji szeregowej możemy rozpocząć wtedy kiedy wykryjemy, że w buforze szeregowym znajdują się jakieś bity danych. Sprawdzenia tego dokonujemy za pomocą instrukcji warunkowej i funkcji `if(Serial.available() > 0)`. Jeśli ten warunek jest spełniony to możemy rozpocząć pobieranie danych.

W ćwiczeniu 4 przesyłane dane były typu łańcuchowego `string`. Tzn., w programie rozpatrywane były jako znaki tekstowe. Użyliśmy do tego polecenia `Serial.readStringUntil('\n')`. Jednak kiedy byśmy chcieli przez transmisję szeregową przesłać dane liczbowe do obliczeń musielibyśmy odebrać je inną instrukcją. Dla znanej nam zmiennej typu całkowitego instrukcja ta wygląda następująco:

```
int liczba;
liczba=Serial.parseInt();
```

4. Ćwiczenie 1. Do omówienia przez nauczyciela.

Napisz program, który umożliwi przesłanie z komputera dwóch liczb typu całkowitego różne od zera. Następnie odeśle w odpowiedzi UART informację o sumie, różnicy i iloczynie tych liczb.

Kod źródłowy rozwiązania ćwiczenia:

```
int a=0;
int b=0;

void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
```

```
  Serial.begin(9600);
}

void loop() {

  if(Serial.available() > 0 && a==0)
  {
    a=Serial.parseInt();
    if(a==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }

  if(Serial.available() > 0 && a!=0 && b==0)
  {
    b=Serial.parseInt();
    if(b==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }

  if(a!=0 && b!=0)
  {
    string wynik=(string)a+'+'+(string)b+'='+(string)(a+b);
    Serial.println(wynik);
    wynik=(string)a+'-'+(string)b+'='+(string)(a-b);
    Serial.println(wynik);
    wynik=(string)a+'*'+(string)b+'='+(string)(a*b);
    Serial.println(wynik);
    a=0;
    b=0;
  }

}
```

W programie po ustawieniu prędkości transmisji sprawdzany jest stan bufora transmisji szeregowej oraz wartość liczby `a`. Jeśli w buforze oczekują jakieś znaki i wartość `a` równa jest zeru, to do zmiennej `a` przypisujemy wartość poleceniem `a=Serial.parseInt();` . Gdyby odebrana wartość była ponownie zerowa, do użytkownika zostanie wysłany komunikat o błędzie.

Podobnie postępujemy z liczbą `b` – w tym przypadku w warunku logicznym sprawdzamy również, czy liczba `a` jest już różna od zera.

Jeśli obie liczby `a` i `b` są podane, możemy wyświetlić wynik. Komunikat o wynikach działania w przykładowym programie został wykonany następująco. Utworzono zmienną `stringwynik`. W przypisaniu do tej zmiennej dokonano połączenia za pomocą znaku + kilku łańcuchów tek-

stu. Wartości zmiennej a, znaku operacji, wartości zmiennej b, znaku '=' oraz wartości wynikowej.

Ponieważ zmienna `a` jest liczbą, a nie łańcuchem znaków, musieliśmy ją przekonwertować na `string`. Taka operacja nosi nazwę **rzutowania** na typ danych. W tym przypadku rzutując na zmienną typu `string` wystarczy przed nazwą zmiennej liczbowej wpisać w nawiasie zwykłym typ, na który rzutujemy daną liczbę. Np. `(string)a`.

rzutowanie

5. Omówienie przez nauczyciela, czym jest biblioteka w języku C++.

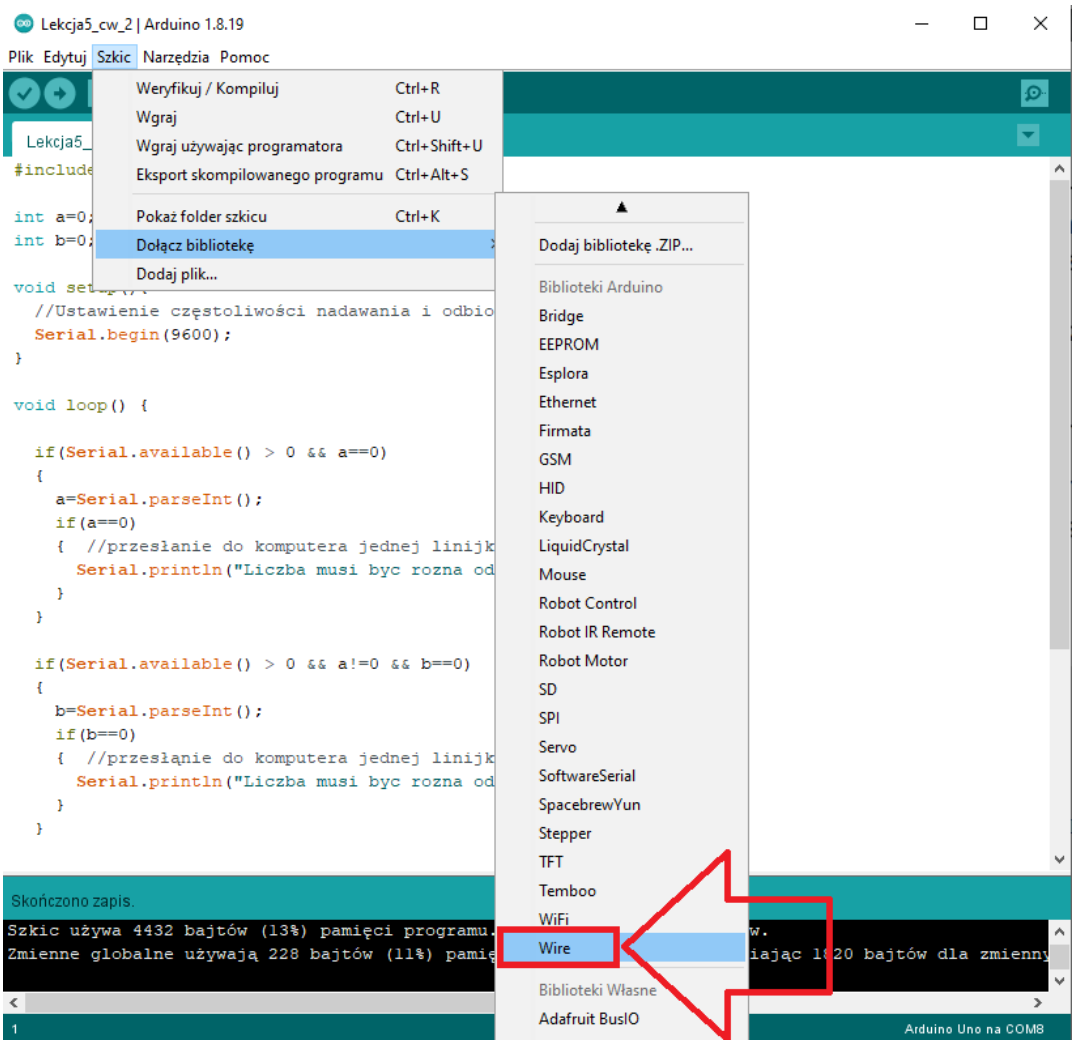
W języku C++ dla Arduino tylko podstawowe polecenia wbudowane są w kompilator. Wiąże się to z tym, że jeśli chcemy skorzystać z dołączonych do układu Arduino UNO dodatkowych peryferii, często musimy korzystać z poleceń nieznanych kompilatorowi. W takiej sytuacji musimy dołączyć do naszego programu dodatkowy plik, w którym zwarte są funkcje, jakich chcemy użyć. Taki plik będziemy nazywać **biblioteką**. Biblioteka dołączona do naszego programu będzie swoistym „słownikiem", w którym kompilator w trakcie analizy naszego programu będzie mógł odszukać używane niestandardowe funkcje. Biblioteki dołączamy do programu za pomocą dyrektywy `#include <nazwa_biblioteki.h>`.

biblioteka

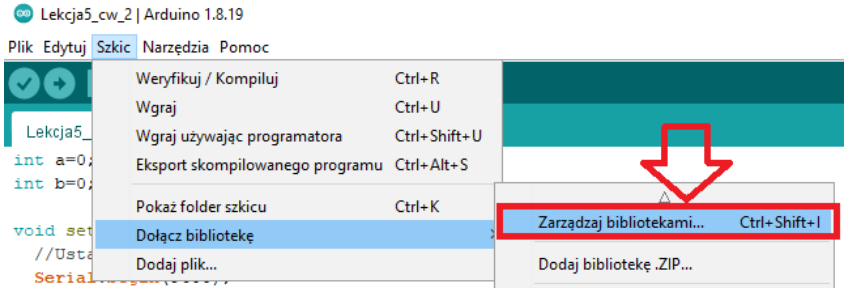
6. Instalacja biblioteki wyświetlacza LCD

Do pracy z wyświetlaczem LCD wbudowanym w zestaw edukacyjny TME-EDU-ARD-2. potrzebujemy trzech bibliotek. Są to: `Wire.h`, `hd44780.h`, `hd44780ioClass/hd44780_I2Cexp.h`.

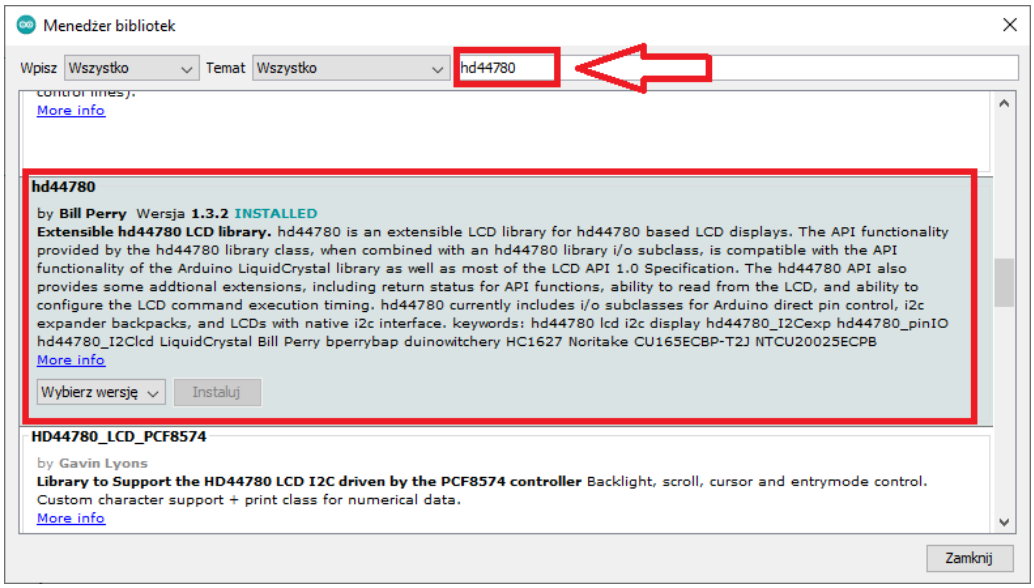
Zanim dołączymy je do naszego projektu, musimy się upewnić, czy mamy je pobrane i czy są one zainstalowane w naszym środowisku Arduino IDE. W tym celu musimy wejść w Menu główne Szkic -> Dołącz bibliotekę. Biblioteka `Wire.h` jest biblioteką Arduino UNO i powinna już być zainstalowana i widoczna w wykazie bibliotek.



Dla bibliotek `hd44780.h` oraz `hd44780ioClass/hd44780_I2Cexp.h` konieczne będzie prawdopodobnie doinstalowanie całego pakietu bibliotek do układu `hd44780.h`. W tym celu wchodzimy w opcję Zarządzaj bibliotekami (dostępną przy wykazie bibliotek).



Aby odnaleźć odpowiedni pakiet bibliotek, wpisujemy w polu wyszukiwania „hd44780” i na uzyskanej liście odszukujemy i instalujemy `hd44780`.



7. Ćwiczenie 2. Do omówienia przez nauczyciela.

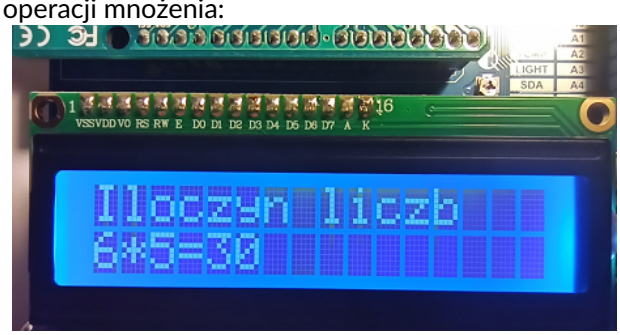
Napisz program, który umożliwi przesłanie z komputera dwóch liczb typu całkowitego różnych od zera. Następnie obliczy sumę, różnicę i iloczyn tych liczb i wyświetli na wyświetlaczu LCD.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

int a=0;
int b=0;

void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  if(Serial.available() > 0 && a==0)
  {
    a=Serial.parseInt();
    if(a==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }
  if(Serial.available() > 0 && a!=0 && b==0)
```

```
    {
      b=Serial.parseInt();
      if(b==0)
      {  // przesłanie do komputera jednej linijki tekstu
        Serial.println("Liczba musi byc rozna od 0");
      }
    }

    if(a!=0 && b!=0)
    {
      string wynik=(string)a+'+'+(string)b+'='+(string)(a+b);
       // ustaw pozycję kursora na lewy górny róg (zerowy wiersz)
      lcd.setCursor(0, 0);
      // wypisz na wyświetlaczu wykonywane działanie
      lcd.print("Suma liczb");
      // ustaw pozycję kursora na drugi wiersz
      lcd.setCursor(0, 1);
      // wypisz wynik
      lcd.print(wynik);
      delay(1000);
      lcd.clear();

      wynik=(string)a+'-'+(string)b+'='+(string)(a-b);


      lcd.setCursor(0, 0);
      lcd.print("Roznica liczb");
      lcd.setCursor(0, 1);
      lcd.print(wynik);
      delay(1000);
      lcd.clear();

      wynik=(string)a+'*'+(string)b+'='+(string)(a*b);

      lcd.setCursor(0, 0);
      lcd.print("Iloczyn liczb");
      lcd.setCursor(0, 1);
      lcd.print(wynik);
      delay(1000);
      lcd.clear();
      a=0;
      b=0;
    }
  }
```

Przykładowy efekt działania programu dla wyświetlenia operacji mnożenia:



W programie tym do przypisania zmiennym danych i wykonania operacji wykorzystano taki sam algorytm jak w ćwiczeniu 1. Również w tym przykładzie utworzono zmienną łańcuchową `wynik`, do której przypisano łańcuch znakowy z działaniem matematycznym i jego wynikiem.

Wyświetlenie wyniku dla każdej operacji zostało zrealizowane jako osobne odsłony napisów wyświetlonych na wyświetlaczu LCD. Czas wyświetlenia jednego wyniku działania to 1 sekunda. Do wyświetlenia wyników na wyświetlaczu LCD użyto dodatkowych bibliotek:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
```

Następnie wykonano konieczne konfiguracje wyświetlacza za pomocą instrukcji:

```
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);
lcd.begin(16, 2);
```

Pierwsza z instrukcji konfiguracyjnych określa m.in., na jakich wyprowadzeniach (PIN-ach) układu Arduino UNO jest podłączony wyświetlacz LCD w zestawie edukacyjnym TME-EDU-ARD-2.

Druga określa rodzaj wyświetlacza. Tu – wyświetlacz LCD po 16 znaków w 2 rzędach.

Poleceniem `lcd.setCursor(0, 0);` określono pozycję w której ma rozpocząć pisanie program. Pozycja 0,0 określa zerowy rząd i zerową kolumnę. Czyli w praktyce lewy górny róg wyświetlacza.

Polecenie `lcd.print("Suma liczb");` wysyła do wyświetlacza żądany napis. Napis ten będzie się rozpoczynał w pozycji wskazanej przez `lcd.setCursor(0, 0);`.

Ostatnie polecenie użyte do wyświetlacza LCD w programie to `lcd.clear()`. Dzięki niemu ekran jest czyszczony ze znaków i możemy rozpocząć wyświetlanie kolejnych informacji.

8. Omówienie przez nauczyciela zmiennej rzeczywistej `float`.

W kalkulatorze, który stworzyliśmy w oparciu o Arduino UNO, nie wykonaliśmy jeszcze operacji dzielenia. Wynik tej operacji zwykle nie będzie liczbą całkowitą, ale składającą się z części całkowitej i ułamkowej. Do reprezentacji takich liczb służą typy zwane rzeczywistymi lub zmiennoprzecinkowymi, `float` i `double`. Zmienna `float` potrafi przechować 6–7 (dziesiętnych) cyfr znaczących. Jest to łączna liczba cyfr w części całkowitej i ułamkowej. Jeśli nasza dana ma w zapisie więcej cyfr znaczących niż możliwości zmiennej `float`, to jest zaokrąglana do możliwości `float` 'a. Zmienna `double` ma możliwość przechowania 2 razy większej ilości cyfr znaczących.

Dla zaawansowanych: Zmienna typu `float` ma 4 bajty, z czego jeden to cecha (wykładnik, −127..128), zaś pozostałe trzy to bajty mantysy, w tym jeden bit znaku.
(Szczegóły np. na: `https://docs.microsoft.com/en-us/cpp/c-language/type-float?view=msvc-170`)

Przykładowa deklaracja zmiennej float:

```
float liczba=2.34
```

9. Ćwiczenie 3. Do omówienia przez nauczyciela.

Napisz program, który podobnie jak ćwiczeniu 1 umożliwi przesłanie z komputera dwóch liczb, różnych od zera, ale typu rzeczywistego. Następnie wypisze sumę, różnicę, iloczyn i iloraz tych liczb na wyświetlaczu LCD.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

float a=0;
float b=0;
void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  if(Serial.available() > 0 && a==0)
  {
    a=Serial.parseFloat();
    if(a==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }

  if(Serial.available() > 0 && a!=0 && b==0)
  {
    b=Serial.parseFloat();
    if(b==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
```

```
    }
    if(a!=0 && b!=0)
    {
      string wynik=(string)a+'+'+(string)b+'='+(string)(a+b);
       // ustaw pozycję kursora na lewy górny róg (zerowy wiersz)
      lcd.setCursor(0, 0);
      // wypisz na wyświetlaczu wykonywane działanie
      lcd.print("Suma liczb");
      // ustaw pozycję kursora na drugi wiersz
      lcd.setCursor(0, 1);
      // wypisz wynik
      lcd.print(wynik);
      delay(3000);
      lcd.clear();

      wynik=(string)a+'-'+(string)b+'='+(string)(a-b);

      lcd.setCursor(0, 0);
      lcd.print("Roznica liczb");
      lcd.setCursor(0, 1);
      lcd.print(wynik);
      delay(3000);
      lcd.clear();

      wynik=(string)a+'*'+(string)b+'='+(string)(a*b);

      lcd.setCursor(0, 0);
      lcd.print("Iloczyn liczb");
      lcd.setCursor(0, 1);
      lcd.print(wynik);
      delay(3000);
      lcd.clear();

      wynik=(string)a+'/'+(string)b+'='+(string)(a/b);

      lcd.setCursor(0, 0);
      lcd.print("Iloraz liczb");
      lcd.setCursor(0, 1);
      lcd.print(wynik);
      delay(3000);
      lcd.clear();
      a=0;
      b=0;
    }
  }
```

Program w porównaniu z ćwiczeniem 2 ma zmieniony typ zmiennych `a` i `b` na `float` co uwzględniono w ich deklaracji. Druga zmiana dotyczy sposobu pobrania danych od użytkow-

nika. Instrukcja, która poprawnie pobiera dane zmiennoprzecinkowe z UART to `Serial.parseFloat();`.

Ostatnia zmiana to dodanie czwartego działania wyświetlanego na wyświetlaczu LCD: dzielenia.

10. Ćwiczenie 4. Do omówienia przez nauczyciela.

Napisz program, który będzie pobierał niezerowe liczby od użytkownika przez interfejs UART. Po pobraniu każdej liczby będzie ona wyświetla na wyświetlaczu LCD. Kiedy użytkownik wciśnie przycisk monostabilny SW_CENTER Arduino UNO powinno obliczyć średnią arytmetyczną z podanych liczb i wypisać jej wartość na wyświetlaczu LCD.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_CENTER 8

float suma=0;
float liczba=0;
int ilosc=0;

void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  if(Serial.available() > 0)
  {
    liczba=Serial.parseFloat();
    if( liczba!=0)
    {
      lcd.clear();
      suma+=liczba;
      ilosc++;
      lcd.setCursor(0, 0);
      lcd.print(liczba);
      lcd.setCursor(0, 1);
      lcd.print(ilosc);
    }
  }
  if (digitalRead(SW_CENTER) == HIGH)
```

```
    {
      lcd.setCursor(0, 0);
      lcd.print("Srednia wynosi");
      lcd.setCursor(0, 1);
      lcd.print(suma/ilosc);
    }
  }
```

W programie wykorzystano trzy zmienne. Zmienna `suma` służy do przechowywania sumy podanych przez użytkownika liczb. Zmienna `liczba` służąca przechowywaniu bieżącej liczby pobranej od użytkownika. Zmienna `ilosc`, w której będziemy zliczać ilość liczb które podał użytkownik. Wszystkie te zmienne na początku wyzerowaliśmy by mieć pewność, że nie znajdują się w nich żadne niepożądane wartości.

Jeżeli w buforze transmisji szeregowej wykryjemy, że oczekują jakieś dane do odebrania, to pobieramy je do zmiennej `liczba` poleceniem `liczba=Serial.parseFloat()` . Następnie sprawdzamy, czy podana wartość jest niezerowa i czyścimy ekran, a następnie wyświetlamy ją w pierwszej linii wyświetlacza LCD. W drugiej linii wyświetlamy ilość podanych do tej pory liczb do średniej arytmetycznej. Następnie pobraną liczbę dodajemy do sumy (do zmiennej `suma`).

Jeżeli wykryjemy stan wysoki na przycisku `SW_CENTER`, czyścimy wyświetlacz LCD z danych i wyświetlamy obliczony wynik średniej arytmetycznej.

11. Przeprowadzenie quizu podsumowującego lekcję.

1. Bibliotekę instrukcji do programu C++ dołączymy za pomocą:
   a) #include
   b) #define
   c) <include>
   d) <define>
2. Instrukcja lcd.begin(16, 2);
   a) Konfiguruje wyświetlacz
   b) Wypisuje napis początkowy
   c) Zmienia kontrast
   d) Ustawia kursor na początku ekranu
3. Wyświetlacz LCD wyczyścisz ze znaków poleceniem:
   a) lcd.setCursor(0, 0);
   b) lcd.print("");
   c) lcd.clear();
   d) lcd.print(clear);
4. Polecenie lcd.setCursor(0, 0); ustawia kursor
   a) W lewym górnym rogu wyświetlacza
   b) W prawym górnym rogu wyświetlacza
   c) W lewym dolnym rogu wyświetlacza
   d) W prawym dolnym rogu wyświetlacza
5. Zmienna float(zaznacz prawdziwe)
   a) Nie wymaga deklaracji

b) Jest taką samą zmienną jak zmienna int
c) Przechowuje zawsze 7 cyfr po przecinku
d) Separatorem części ułamkowej liczby zapisanej w zmiennej float jest kropka.

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który umożliwi przesłanie z komputera dwóch liczb typu całkowitego różne od zera. Następnie odeśle w odpowiedzi UART informację o sumie, różnicy i iloczynie tych liczb.

Kod źródłowy rozwiązania ćwiczenia:

```
int a=0;
int b=0;

void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
}

void loop() {

  if(Serial.available() > 0 && a==0)
  {
    a=Serial.parseInt();
    if(a==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }

  if(Serial.available() > 0 && a!=0 && b==0)
  {
    b=Serial.parseInt();
    if(b==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }

  if(a!=0 && b!=0)
  {
    string wynik=(string)a+'+'+(string)b+'='+(string)(a+b);
    Serial.println(wynik);
    wynik=(string)a+'-'+(string)b+'='+(string)(a-b);
    Serial.println(wynik);
```

```
    wynik=(string)a+'*'+(string)b+'='+(string)(a*b);
    Serial.println(wynik);
    a=0;
    b=0;
  }
}
```

Ćwiczenie 2

Napisz program, który umożliwi przesłanie z komputera dwóch liczb typu całkowitego różnych od zera. Następnie obliczy sumę, różnicę i iloczyn tych liczb i wyświetli na wyświetlaczu LCD.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

int a=0;
int b=0;

void setup(){
  // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  if(Serial.available() > 0 && a==0)
  {
    a=Serial.parseInt();
    if(a==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }
  if(Serial.available() > 0 && a!=0 && b==0)
  {
    b=Serial.parseInt();
    if(b==0)
    {  // przesłanie do komputera jednej linijki tekstu
      Serial.println("Liczba musi byc rozna od 0");
    }
  }
```

```
    if(a!=0 && b!=0)
    {
      string wynik=(string)a+'+'+(string)b+'='+(string)(a+b);
       // ustaw pozycję kursora na lewy górny róg (zerowy wiersz)
      lcd.setCursor(0, 0);
      // wypisz na wyświetlaczu wykonywane działanie
      lcd.print("Suma liczb");
      // ustaw pozycję kursora na drugi wiersz
      lcd.setCursor(0, 1);
      // wypisz wynik
      lcd.print(wynik);
      delay(1000);
      lcd.clear();

      wynik=(string)a+'-'+(string)b+'='+(string)(a-b);


      lcd.setCursor(0, 0);
      lcd.print("Roznica liczb");
      lcd.setCursor(0, 1);
      lcd.print(wynik);
      delay(1000);
      lcd.clear();

      wynik=(string)a+'*'+(string)b+'='+(string)(a*b);

      lcd.setCursor(0, 0);
      lcd.print("Iloczyn liczb");
      lcd.setCursor(0, 1);
      lcd.print(wynik);
      delay(1000);
      lcd.clear();
      a=0;
      b=0;
    }
}
```

Ćwiczenie 3

Napisz program, który podobnie jak ćwiczeniu 1 umożliwi przesłanie z komputera dwóch liczb, różnych od zera, ale typu rzeczywistego. Następnie wypisze sumę, różnicę, iloczyn i iloraz tych liczb na wyświetlaczu LCD.

Ćwiczenie 4

Napisz program, który będzie pobierał niezerowe liczby od użytkownika przez interfejs UART. Po pobraniu każdej liczby będzie ona wyświetla na wyświetlaczu LCD. Kiedy użytkownik wci-

śnie przycisk monostabilny SW_CENTER Arduino UNO powinno obliczyć średnią arytmetyczną z podanych liczb i wypisać jej wartość na wyświetlaczu LCD.

Quiz

1. Bibliotekę instrukcji do programu C++ dołączymy za pomocą:
   a) #include
   b) #define
   c) <include>
   d) <define>
2. Instrukcja lcd.begin(16, 2);
   a) Konfiguruje wyświetlacz
   b) Wypisuje napis początkowy
   c) Zmienia kontrast
   d) Ustawia kursor na początku ekranu
3. Wyświetlacz LCD wyczyścisz ze znaków poleceniem:
   a) lcd.setCursor(0, 0);
   b) lcd.print("");
   c) lcd.clear();
   d) lcd.print(clear);
4. Polecenie lcd.setCursor(0, 0); ustawia kursor
   a) W lewym górnym rogu wyświetlacza
   b) W prawym górnym rogu wyświetlacza
   c) W lewym dolnym rogu wyświetlacza
   d) W prawym dolnym rogu wyświetlacza
5. Zmienna float(zaznacz prawdziwe)
   a) Nie wymaga deklaracji
   b) Jest taką samą zmienną jak zmienna int
   c) Przechowuje zawsze 7 cyfr po przecinku
   d) Separatorem części ułamkowej liczby zapisanej w zmiennej float jest kropka.

Lekcja 6

Wyświetlacz LED i obliczanie binarnej postaci liczby w zestawie edukacyjnym TME-EDU-ARD-2.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych całkowitych, rzeczywistych, instrukcji warunkowych w języku C++, programowania wyświetlacza LCD. Utrwalenie wiadomości nt. stosowania bibliotek w programach C++. Zapoznanie z działaniem pętli `while()`. Nabycie umiejętności stosowania tablic w języku C++. Nabycie wiedzy i ukształtowanie umiejętności nt. zmiennych logicznych `bool`.

Efekty kształcenia

- Uczeń stosuje transmisję UART do komunikacji układu Arduino z komputerem.
- Uczeń zna pojęcie zmiennej logicznej (typu `bool`).
- Uczeń potrafi stosować zmienne logiczne (typu `bool`).
- Uczeń wie jak i kiedy stosuje się pętle `while`.
- Uczeń stosuje pętle `while()` w swoich programach z Arduino UNO.

Wstęp

W lekcji tej utrwalimy umiejętności posługiwania się przyciskami monostabilnymi, transmisją UART, wyświetlaczem LCD i diodą LED. Praca nad przeliczeniami systemów dziesiętnego i binarnego wymagać będzie od nas również umiejętności obsługi tablic w języku C++, zmiennych logicznych `bool` oraz pętli `while()`. Te nowopoznane funkcjonalności języka C++ będziemy testować w pracy z nowym podzespołem, jakim jest wyświetlacz LED.

Przebieg lekcji

1. Przedstawienie celów lekcji.

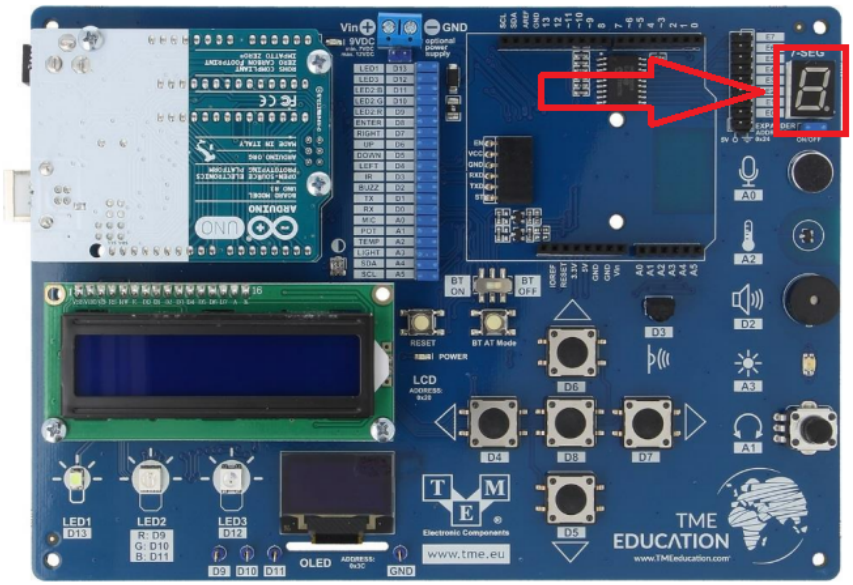
2. Powtórzenie wiadomości z poprzednich lekcji,

1. Do czego służy funkcja `loop()` i `setup()`
2. Jak wgrywać program do układu Arduino
3. Za pomocą jakich podzespołów prowadzona była dotychczas komunikacja z użytkownikiem? Odp. Przyciski monostabilne, dioda LED D1, buzzer.

4. Jak programować poznane na poprzednich lekcjach podzespoły, tj. Przyciski monostabilne, dioda LED D1, buzzer, wyświetlacz LCD?
5. Powtórzenie przez nauczyciela, czym jest UART. Jak należy konfigurować UART w programie Arduino i na komputerze w „Monitorze portu szeregowego"
6. Powtórzenie poznanych typów zmiennych: liczb całkowitych `int`, liczb rzeczywistych `float`, `double`, znaków i łańcuchów znaków `char`, `string`.

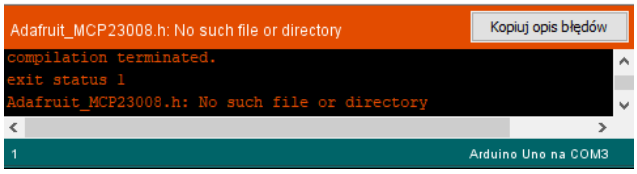
3. Omówienie przez nauczyciela sposobu działania i wyświetlania danych na wyświetlaczu LED.

Wyświetlacz LED zastosowany w zestawie edukacyjnym TME-EDU-ARD-2 jest to mały wyświetlacz 7-segmentowy (tzn. może wyświetlić jeden znak liczbowy). Wyświetlacze LED stosujemy do wyświetlenia cyfr i znaków dających się utworzyć z kształtów 7 niezależnie świecących segmentów diod LED. Zastosowany w naszym zestawie wyświetlacz LED ma tzw. wspólną katodę: styki ujemne diod świecących w poszczególnych segmentach są ze sobą połączone. I zwarte do masy. Sterowanie włączeniem danego segmentu odbywa się przez anody podłączone do kolejnych pinów układu Arduino UNO.

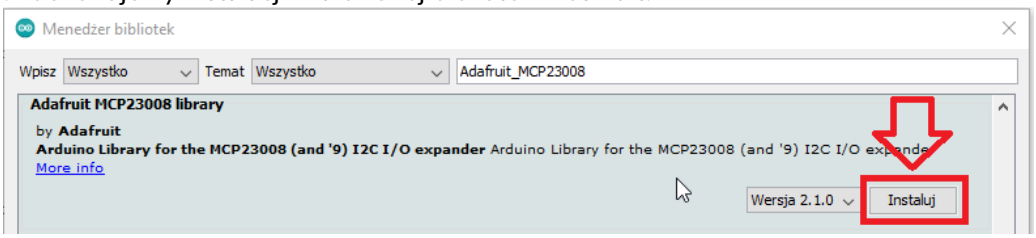


Programowanie wyświetlacza LED jest podobne do programowania diod LED. Musimy na początku programu w funkcji `setup()` dokonać ustawienia kolejnych pinów układu Arduino UNO. Ustawiamy je jako wyjściowe. I dalej obsługujemy poleceniem znanym już wcześniej tj. np. `digitalWrite(5, HIGH)` lub `digitalWrite(5, LOW)`. pierwszy z nich włączy dany segment 5, drugie zaś wyłączy świecenie segmentu 5. W ten sposób możemy konfigurować wzory kształtów znaków. Jest to dość żmudne zadanie, dlatego ogromnym ułatwieniem przy wyświetlaniu danych jest zastosowania dedykowanej diodom LED biblioteki. Jest to Adafruit_MCP23008.h. Zastosowanie tej biblioteki i utworzenie na początku programu obiektu reprezentującego wyświetlacz LED (polecenie `Adafruit_MCP23008 seg7`), pozwoli nam wyświetlać cyfry za pomocą jednego polecenia. Polecenie `wyswietlaczLED.writeGPIO(digit[i])` samo sprawdzi, jakie segmenty muszą być włączone, a jakie wyłączone dla danej cyfry.

Jeśli w trakcie kompilacji programu zobaczysz poniższy komunikat o błędzie, oznacza to, że musisz doinstalować bibliotekę Adafruit_MCP23008.h.



Instalacja biblioteki odbywa się w menedżerze bibliotek który wywołamy w menu głównym Szkic->Dołącz bibliotekę->Zarządzaj bibliotekami. W polu wyszukiwania wpisujemy frazę to Adafruit_MCP23008. I dokonujemy instalacji znalezionej biblioteki Adafruit.



4. Ćwiczenie 1

Napisz program, który po uruchomieniu na wyświetlaczu LED wyświetli wartość 0. W chwili, kiedy użytkownik wciśnie przycisk monostabilny `SW_UP`, wartość wyświetlana powinna zwiększyć się o jeden. Kiedy naciśnięty będzie przycisk `SW_DOWN`, to wartość powinna się pomniejszyć o 1. Program powinien uniemożliwiać przekroczenie wartości skrajnych. Minimalna wyświetlana wartość to 0, maksymalna to 9.

Kod źródłowy rozwiązania ćwiczenia:

```
#include "Adafruit_MCP23008.h"

#define SW_UP 6
#define SW_DOWN 5

Adafruit_MCP23008 seg7;
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
 // konfiguracja pinów ekspandera
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
```

```
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
}
int i=0;

void loop() {
 if (digitalRead(SW_UP) == HIGH && i<9) {
  i++;
 }
 if (digitalRead(SW_DOWN) == HIGH && i>0) {
  i--;
 }

 seg7.writeGPIO(digit[i]);
 delay(500);
}
```

W programie utworzono obiekt o nazwie `seg7` klasy `Adafruit_MCP23008`. By nie rozwijać na tym etapie pojęcia klasy i obiektu zapamiętajmy, że w praktyce utworzony obiekt `seg7` jest to rodzaj zmiennej. Odwołując się do niej możemy zarządzać wyświetlaczem LED.

Do pracy wyświetlacza i reprezentacji na nim cyfr potrzebujemy jeszcze konfiguracji segmentów, które włączą nam odpowiednie segmenty LED prezentujące poszczególne cyfry. Taką konfigurację wykonujemy w tablicy `uint8_t digit[10]`.

Tablica w języku C++ jest to nic innego jak zbiór danych tego samego typu, ponumerowanych (indeksowanych) kolejnymi liczbami naturalnymi. W języku C++ indeksy tablic zaczynają się od 0.

Dla dociekliwych: Ale czemu od 0, a nie od 1? Indeks elementu (danej) w tablicy to offset względem adresu pierwszej danej – aby obliczyć adres danej (komórki tablicy) o indeksie `i`, należy pomnożyć `i` przez liczbę bajtów, jaka przypada na typ danych zadeklarowany dla tablicy. Pierwsza komórka tablicy nie wymaga przesuwania „czytnika”, czyli ma offset równy 0.

W naszej konkretnej deklaracji do tablicy `digit` będziemy przypisywać 10 wartości od 0 do 255 czyli wartości, które w zapisie binarnym będą zapisane na 8 bitach. Przypisanie konfiguracji segmentów to tak naprawdę przypisanie 1 (jedynek) dla segmentów, które mają dla danej cyfry być włączone. Następnie przypisanie 0 (zer) dla segmentów, które dla danej cyfry mają być wyłączone. Całą konfigurację zapisujemy binarnie, dlatego w przypisaniu wartości do tablicy `digit` występuje sekwencja 0 i 1 po literze „B”. Np. dla wartości 0 jest to „`B00111111`”.

W funkcji `void setup()` dokonano konfiguracji pinów odpowiedzialnych za wyświetlanie na poszczególnych segmentach LED.

W programie jako zmienną globalną (czyli zmienną „widoczną” we wszystkich funkcjach programu) zastosowano zmienną typu całkowitego int i. Będzie ona przechowywała aktualnie wyświetlaną cyfrę.

W funkcji `void loop()` sprawdzamy, czy został wciśnięty przycisk do góry lub na dół. Jeśli tak, to w warunku logicznym sprawdzamy również, czy zwiększając lub zmniejszając wartość na wyświetlaczu nie spowodujemy przekroczenia wartości skrajnych (0 lub 9). Jeśli nie, to dokonujemy zmiany zmiennej `i`.

Po tym następuje wyświetlenie cyfry poleceniem `seg7.writeGPIO(digit[i]);`.

5. Omówienie przez nauczyciela procesu obliczenia reprezentacji binarnej danej liczby.

Aby znaleźć binarny zapis danej liczby, należy dokonać jej dzielenia z resztą przez 2. Część całkowitą wyniku zapisujemy do kolejnego działania dzielenia, zaś resztę spisujemy obok wykonanego działania.

Wyżej opisaną operację powtarzamy na kolejnych częściach całkowitych kolejnych dzieleń z resztą, spisując reszty obok. Działania te wykonujemy do momentu, w którym uzyskamy część całkowitą 0.

Przykład – obliczenie binarnej postaci liczby 173.

Liczba dziesiętna/wynik z dzielenia	Działanie	Reszta
wartość do przeliczenia: 173	/2	1
86	/2	0
43	/2	1
21	/2	1
10	/2	0
5	/2	1
2	/2	0
1	/2	1
0	(koniec działań)	

Wynikową postać binarną liczby uzyskujemy spisując reszty z kolejnych dzieleń, ale od ostatniej do pierwszej, czyli od końca.

173_{10} == 10101101_2, liczba 173 (w zwykłym zapisie dziesiętnym) to 10101101 w zapisie binarnym.

6. Omówienie przez nauczyciela pętli `while()`

Pętla `while()` służy w programowaniu do powtarzania danej sekwencji instrukcji tak długo, jak długo warunek logiczny pętli jest spełniony (ang. „while” – ‘podczas gdy’).

Pętla `while()` ma następującą strukturę:

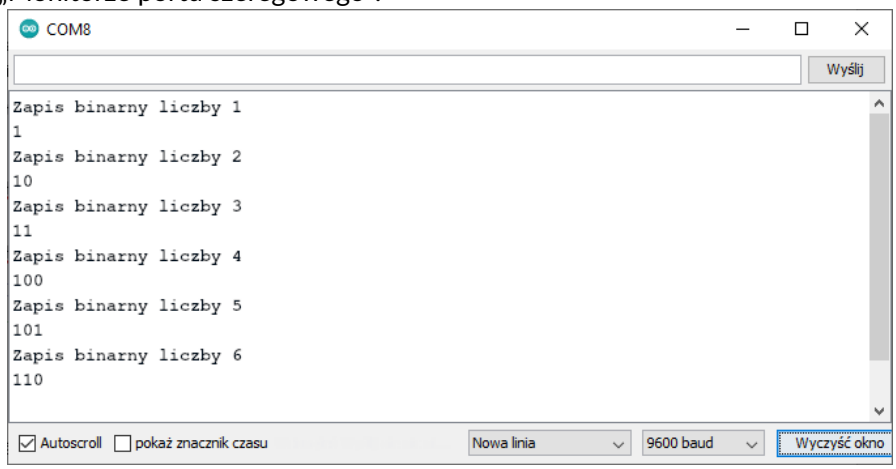
```
while(⟨warunek logiczny⟩)
{
    // instrukcje powtarzane w pętli
}
```

W pętli `while()` warunek logiczny sprawdzany jest przed wykonaniem sekwencji instrukcji. Dlatego jeśli jest on od razu fałszywy, instrukcje znajdujące się w funkcji zostaną pominięte.

7. Ćwiczenie 2

Napisz program, który obliczy postać binarną liczby wskazywanej (w postaci dziesiętnej) przez wyświetlacz LED z poprzedniego ćwiczenia. Wynik prześlij przez interfejs UART.

Przykładowy efekt widoczny na „Monitorze portu szeregowego”:



Kod źródłowy rozwiązania ćwiczenia:

```
#include "Adafruit_MCP23008.h"

#define SW_UP 6
#define SW_DOWN 5

Adafruit_MCP23008 seg7;
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
 // konfiguracja pinów ekspandera
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
```

```
 seg7.pinMode(7, OUTPUT);
 // Ustawienie częstoliwości nadawania i odbioru
 Serial.begin(9600);
}

int i=0;
bool binarna[8]={0,0,0,0,0,0,0,0};
int liczba=0;// zmienna pomocnicza do przechowanie dzielonej wartości
void loop() {

 if (digitalRead(SW_UP) == HIGH && i<9) {
  i++;
 }
 if (digitalRead(SW_DOWN) == HIGH && i>0) {
  i--;
 }
 seg7.writeGPIO(digit[i]);
 if(i!=liczba)
 {
   liczba=i;// zmienna pomocnicza do przechowanie dzielonej wartości
   int j=7;// zmienna pomocnicza zliczająca, którą z kolei resztę z dzielenia zapisujemy
   while(liczba>0)
   {
    binarna[j]=liczba%2;
    liczba=liczba/2;
    j--;
   }

   Serial.print("Postać binarna liczby ");
   Serial.println(i);

   while(j!=7)
   {
    j++;
            Serial.print(binarna[j]);

   }
   Serial.println("");
    liczba=i;
 }
 delay(500);
}
```

Do realizacji programu zadeklarowaliśmy tablicę zmiennych logicznych `bool binarna[8]`. Tablica ta ma rozmiar 8. Co oznacza, że może przechować 8 wartości typu logicznego (0,1). Chcąc mieć pewność, że w momencie deklaracji w tablicy nie znalazły się jakieś niepożądane wartości, wszystkim jej komórkom przypisujemy wartość 0 (zero, fałsz).

Zmienna całkowita `i` służy przechowaniu liczby, która jest wyświetlana na wyświetlaczu LED

(w postaci dziesiętnej), i której postać binarną obliczymy. Przyciski `SW_UP` i `SW_DOWN` modyfikują wartość zmiennej `i`.

Pojawienie się zmiany w wartości liczby wejściowej, podlegającej przeliczeniu, kontroluje funkcja `if(i!=liczba)`. Bowiem zmienna całkowita `liczba` przechowuje ostatnio przeliczaną wartość. Więc jeśli liczba jest różna od `i`, to znaczy, że jest coś nowego do przeliczenia.

W instrukcjach odpowiedzialnych za obliczenie postaci binarnej przypisujemy na początku do zmiennej pomocniczej `liczba` wartość `i`. Robimy tak, ponieważ potrzebujemy zmiennej, którą będziemy mogli dzielić przez 2 i tym samym modyfikować jej wartość. Nie chcemy zaś nadpisać wartości wejściowej – tej, którą przeliczamy, i którą przechowujemy w zmiennej `i`.

Obliczenie postaci binarnej odbywa się przez zmiany wartości zmiennej `liczba` o połowę, aż do osiągnięcia 0. Dlatego właśnie taki warunek logiczny zastosowaliśmy w naszej pętli `while(liczba>0)`.

W pętli tej musimy spisywać reszty z dzielenia przez 2. To one będą stanowić naszą odpowiedź do zadania. Reszty te spisujemy do tablicy `binarna[]`. Dla późniejszej łatwości odczytu wyniku wpisywanie wyników (reszt z dzielenia przez 2) rozpoczynamy od końca tablicy czyli od `j=7`.

Na koniec w pętli `while(j!=7)` wysyłamy „bit po bicie” wynik z tablicy `binarna`. Tym razem możemy już odczytywać wynik w kierunku rosnącym.

Tak jak mówiliśmy wcześniej, chcemy, by zmienna `liczba` przechowywała po obliczeniu postaci binarnej wartość tej ostatnio przeliczonej liczby. Dlatego po obliczeniu postaci binarnej ponownie przypisujemy na zmienną `liczba` wartość `i`.

8. Ćwiczenie 3

Napisz program, który obliczy postać binarną liczby wskazywanej przez wyświetlacz LCD (w postaci dziesiętnej). Wynik prześlij na wyświetlacz LCD. Postać dziesiętną będziemy wyświetlać w pierwszej linii wyświetlacza LCD. W drugiej linii powinna pojawiać się każdorazowo postać binarna. Wartość wejściowa (w postaci dziesiętej) powinna być regulowana przez przyciski monostabilne góra/dół w zakresie od 0 do 65535.

Przykładowy widok rozwiązania na wyświetlaczu LCD:

W programie zastosowano znaną z ćwiczenia 2 procedurę przeliczającą liczbę dziesiętną na binarną. Wynik został zapisany w zmiennej tablicowej przechowującej typ logiczny 0/1.

Wysłanie wyniku zrealizowano tu w pętli `while(j!=15)`, gdzie dla wyświetlenia poszczególnych bitów ustawiany jest kursor w odpowiedniej pozycji drugiego rzędu wyświetlacza LCD, a następnie drukowany jest bit wyniku z tablicy `lcd.print(binarna[j]);`.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_UP 6
#define SW_DOWN 5

void setup() {
// konfiguracja LCD
  lcd.begin(16, 2);

}
int i=0;
bool binarna[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int liczba=0;// zmienna pomocnicza do przechowanie dzielonej wartości
void loop() {

 if (digitalRead(SW_UP) == HIGH && i<65535) {
  i++;
 }
 if (digitalRead(SW_DOWN) == HIGH && i>0) {
  i--;
 }
 if(i!=liczba)
 {
   liczba=i;// zmienna pomocnicza do przechowanie dzielonej wartości
   int j=15;// zmienna pomocnicza zliczająca, którą z kolei resztę z dzielenia zapisujemy
   while(liczba>0)
   {
    binarna[j]=liczba%2;
    liczba=liczba/2;
    j--;
   }
  lcd.clear();
  lcd.setCursor(0, 0);
  // wypisz na wyświetlaczu wykonywane działanie
  lcd.print(i);
   while(j!=15)
   {
    j++;
```

```
      // ustaw pozycję kursora na drugi wiersz i j-oty znak w wierszu
      lcd.setCursor(j, 1);
      // wypisz wynik
      lcd.print(binarna[j]);

    }

     liczba=i;
  }
  delay(500);
}
```

9. Ćwiczenie 4. **Dla zaawansowanych.**

Zmodyfikuj program z ćwiczenia 3 tak, aby można było wyzerować liczbę dziesiętną (wejściową) za pomocą przycisku `SW_CENTER`.

Dodatkowo zmodyfikuj zwiększanie i zmniejszanie wartości tak, by liczba wejściowa (w postaci dziesiętnej) zmieniała się o wartość wskazywaną jako rząd wielkości na wyświetlaczu LED.

Przykładowo:

- na wyświetlaczu LED wyświetla się „0” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 1;
- na wyświetlaczu LED wyświetla się „1” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 10
- na wyświetlaczu LED wyświetla się „2” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 100
- na wyświetlaczu LED wyświetla się „3” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 1000
- na wyświetlaczu LED wyświetla się „4” przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 10 000

Wartość na wyświetlaczu LED reguluj za pomocą przycisków `SW_LEFT` i `SW_RIGHT`.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <math.h>
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
#include "Adafruit_MCP23008.h"
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_UP 6
#define SW_DOWN 5
#define SW_RIGHT 7
#define SW_CENTER 8
#define SW_LEFT 4
```

```
Adafruit_MCP23008 seg7;
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
  // konfiguracja pinów ekspandera
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
// konfiguracja LCD
  lcd.begin(16, 2);
}
int i=0;
bool binarna[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
int liczba=0;// zmienna pomocnicza do przechowania dzielonej wartości
int mnoznik=0;// zmienna przechowująca rząd wielkości zmian watości przeliczanej na binarną

void loop() {
  seg7.writeGPIO(digit[mnoznik]);
 if (digitalRead(SW_UP) == HIGH && i<65535) {
  i=i+ceil(pow(10,mnoznik));
 }
 if (digitalRead(SW_DOWN) == HIGH && i>0) {
  i=i-ceil(pow(10,mnoznik));
 }
 if (digitalRead(SW_CENTER) == HIGH) {
  i=0;
 }
 if (digitalRead(SW_LEFT) == HIGH && mnoznik>0) {
  mnoznik--;
```

```
  }
  if (digitalRead(SW_RIGHT) == HIGH && mnoznik <4){
   mnoznik++;
  }

  if(i!=liczba)
  {
    liczba=i;// zmienna pomocnicza do przechowanie dzielonej wartości
    int j=15;// zmienna pomocnicza zliczająca, którą z kolei resztę z dzielenia zapisujemy
    while(liczba>0)
    {
     binarna[j]=liczba%2;
     liczba=liczba/2;
     j--;
    }
   lcd.clear();
   lcd.setCursor(0, 0);

   // wypisz na wyświetlaczu wykonywane działanie
   lcd.print(i);
   lcd.setCursor(8, 0);

   char binarnaS[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    while(j!=15)
    {
     j++;
     // ustaw pozycję kursora na drugi wiersz i j-oty znak w wierszu
     lcd.setCursor(j, 1);
     // wypisz wynik
     lcd.print(binarna[j]);
    }

     liczba=i;
  }
  delay(500);
 }
```

W programie do przeliczenia rzędu zmiany wartości zastosowano funkcje `pow()` oraz `ceil()`.

By skorzystać z funkcji `pow()` należy dodatkowo dołączyć do programu bibliotekę `#include <math.h>`. Funkcja `pow(podstawa,wykładnik)` oblicza potęgę liczby podanej w parametrze. Jednak przez niezbyt dokładne wyniki obliczeń tej funkcji musimy zwróconą przez nią wartość potęgi zaokrąglić w górę. Do zaokrąglania w górę używamy funkcji `ceil()`.

10. Przeprowadzenie quizu podsumowującego lekcję.

1. Na wyświetlaczu LED 7-segmentowym po odpowiedniej konfiguracji możemy wyświetlić:

a) tylko cyfry
b) tylko małe litery
c) tylko duże litery
d) cyfry i niektóre litery

2. Tablica `uint8_t digit[10]` używana była do:
   a) Przechowania wartości dziesiętnej liczby wyświetlanej
   b) Przechowania wartości binarnej liczby wyświetlanej
   c) Losowej sekwencji cyfr
   d) Konfiguracji segmentów LED wyświetlanych dla konkretnych cyfr.
3. Funkcja `seg7.writeGPIO(digit[i]);` wyświetla
   a) Liczbę znajdującą się w zmiennej i
   b) Losową liczbę
   c) Losową literę
   d) Instrukcja napisana jest niepoprawnie
4. Liczba 145_{10} to w zapisie binarnym:
   a) 11100000
   b) 10001001
   c) 10010001
   d) 10000001
5. Postać binarna liczby 209_{10} to:
   a) 11000000
   b) 10000011
   c) 11010001
   d) 10001011

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który po uruchomieniu na wyświetlaczu LED wyświetli wartość 0. W chwili, kiedy użytkownik wciśnie przycisk monostabilny `SW_UP`, wartość wyświetlana powinna zwiększyć się o jeden. Kiedy naciśnięty będzie przycisk `SW_DOWN`, to wartość powinna się pomniejszyć o 1. Program powinien uniemożliwiać przekroczenie wartości skrajnych. Minimalna wyświetlana wartość to 0, maksymalna to 9.

Kod źródłowy rozwiązania ćwiczenia:

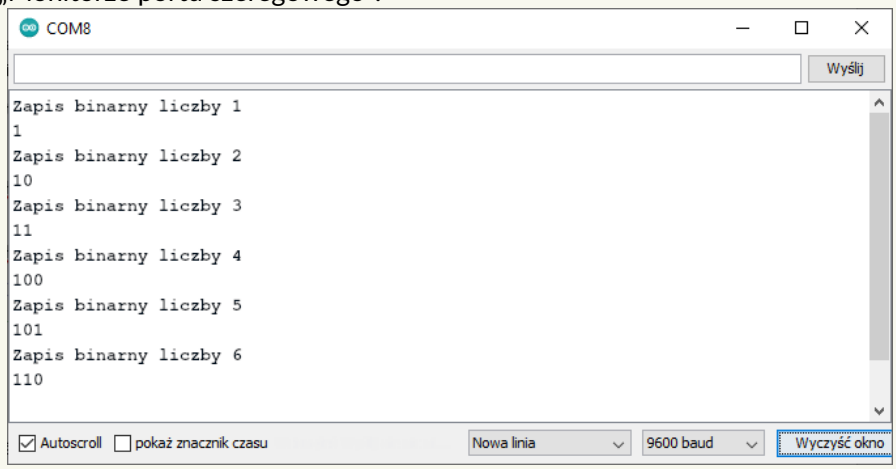
```
#include "Adafruit_MCP23008.h"

#define SW_UP 6
#define SW_DOWN 5

Adafruit_MCP23008 seg7;
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
 // konfiguracja pinów ekspandera
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
}
```

```
int i=0;

void loop() {
 if (digitalRead(SW_UP) == HIGH && i<9) {
  i++;
 }
 if (digitalRead(SW_DOWN) == HIGH && i>0) {
  i--;
 }

 seg7.writeGPIO(digit[i]);
 delay(500);
}
```

Ćwiczenie 2

Napisz program, który obliczy postać binarną liczby wskazywanej (w postaci dziesiętnej) przez wyświetlacz LED z poprzedniego ćwiczenia. Wynik prześlij przez interfejs UART.

Przykładowy efekt widoczny na „Monitorze portu szeregowego”:



Ćwiczenie 3

Napisz program, który obliczy postać binarną liczby wskazywanej przez wyświetlacz LCD (w postaci dziesiętnej). Wynik prześlij na wyświetlacz LCD. Postać dziesiętną będziemy wyświetlać w pierwszej linii wyświetlacza LCD. W drugiej linii powinna pojawiać się każdorazowo postać binarna. Wartość wejściowa (w postaci dziesiętej) powinna być regulowana przez przyciski monostabilne góra/dół w zakresie od 0 do 65535.

Przykładowy widok rozwiązania na wyświetlaczu LCD:

Ćwiczenie 4

Zmodyfikuj program z ćwiczenia 3 tak, aby można było wyzerować liczbę dziesiętną (wejściową) za pomocą przycisku `SW_CENTER`.

Dodatkowo zmodyfikuj zwiększanie i zmniejszanie wartości tak, by liczba wejściowa (w postaci dziesiętnej) zmieniała się o wartość wskazywaną jako rząd wielkości na wyświetlaczu LED.

Przykładowo:

- na wyświetlaczu LED wyświetla się „0” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 1;
- na wyświetlaczu LED wyświetla się „1” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 10
- na wyświetlaczu LED wyświetla się „2” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 100
- na wyświetlaczu LED wyświetla się „3” – przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 1000
- na wyświetlaczu LED wyświetla się „4” przyciski `SW_UP` i `SW_DOWN` zmieniają wartość o 10 000

Wartość na wyświetlaczu LED reguluj za pomocą przycisków `SW_LEFT` i `SW_RIGHT`.

Quiz

1. Na wyświetlaczu LED 7-segmentowym po odpowiedniej konfiguracji możemy wyświetlić:
   a) tylko cyfry
   b) tylko małe litery
   c) tylko duże litery
   d) cyfry i niektóre litery
2. Tablica `uint8_t digit[10]` używana była do:
   a) Przechowania wartości dziesiętnej liczby wyświetlanej
   b) Przechowania wartości binarnej liczby wyświetlanej
   c) Losowej sekwencji cyfr
   d) Konfiguracji segmentów LED wyświetlanych dla konkretnych cyfr.
3. Funkcja `seg7.writeGPIO(digit[i]);` wyświetla
   a) Liczbę znajdującą się w zmiennej i
   b) Losową liczbę
   c) Losową literę
   d) Instrukcja napisana jest niepoprawnie
4. Liczba 145_{10} to w zapisie binarnym:
   a) 11100000
   b) 10001001
   c) 10010001
   d) 10000001
5. Postać binarna liczby 209_{10} to:
   a) 11000000
   b) 10000011
   c) 11010001
   d) 10001011

Lekcja 7

Pętla `for()`. Poznajemy możliwości potencjometru w zestawie edukacyjnym TME-EDU-ARD-2.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych w języku C++. Utrwalenie wiedzy nt. stosowania pętli `while()` i tablic w języku C++. Zapoznanie z działaniem pętli `for()` w języku C++. Nabycie umiejętności odczytu danych z potencjometru w zestawie edukacyjnym TME-EDU-ARD-2.

Efekty kształcenia

- Uczeń stosuje transmisję UART do komunikacji układu Arduino z komputerem.
- Uczeń potrafi stosować zmienne.
- Uczeń wie jak i kiedy stosuje się pętle `for()`.
- Uczeń stosuje pętle `for()` w swoich programach z Arduino UNO.
- Uczeń stosuje w swoich programach potencjometr.

Wstęp

Lekcja ta będzie kontynuacją nauki języka C++ i poznania kolejnej pętli. Po pętli `while()` poznany teraz pętlę `for()`, w której będziemy mogli określić, ile razy się ona wykona. Lekcję od strony sprzętowej uzupełni poznanie nowego komponentu zestawu edukacyjnego jakim jest potencjometr.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

1. Jak programować poznane na poprzednich lekcjach podzespoły, tj.: przyciski monostabilne, dioda LED D1, buzzer, wyświetlacz LCD, wyświetlacz LED.
2. Powtórzenie przez nauczyciela, czym jest pętla `while()`. Kiedy i jak ją stosujemy?
3. Powtórzenie poznanych typów zmiennych liczb całkowitych, liczb rzeczywistych, łańcuchów znaków i zmiennych logicznych.

3. Omówienie przez nauczyciela pętli `for()`

Pętla `for()`, podobnie jak pętla `while()`, służy do wykonywania wielokrotnie tych samych instrukcji znajdujących się w bloku instrukcji pętli. Pętla `while()` powtarza blok instrukcji dopóki określony warunek logiczny jest prawdziwy. Pętla ta z założenia nie ma z góry określonej liczby powtórzeń. My jako jej użytkownicy też nie określaliśmy sztywno tej liczby, uzależniając wyjście z pętli raczej od osiągnięcia określonego skutku.

W pętli `for()` poza warunkiem logicznym wykonania kolejnych powtórzeń pętli zapisuje się warunki początkowe i instrukcje, które mają się wykonać po każdym przejściu pętli. Wszystko to razem umożliwia czytelne i dokładne określenie, ile powtórzeń wykona ta pętla.

Konstrukcja pętli `for()` jest następująca.

```
for(⟨założenia początkowe⟩;⟨warunek logiczny kontynuacji pętli⟩;⟨instrukcja wykon. po każdym obrocie pętli⟩)
{
    // blok instrukcji
}
```

Jak widzimy w nawiasie funkcji for występują aż trzy parametry. Najczęściej w założeniach początkowych dokonujemy deklaracji zmiennych tzw. Iteracyjnych. Są to zmienne, które pomogą nam liczyć kolejne przejścia pętli `for()`. Warunek logiczny w najprostszej postaci musi zawierać wartość maksymalną, do jakiej ma zwiększać się zmienna iteracyjna, lub wartość minimalną, do jakiej zmienna ma się zmniejszać. W ostatnim parametrze najczęściej wskazujemy, jak ma się zmienić zmienna iteracyjna po każdym zakończeniu pętli., np. zwiększyć się o 1 lub więcej. A może zmniejszyć się o 1? Możemy również w miarę potrzeb wykonywać bardziej skomplikowane operacje arytmetyczne.

Przykład pętli `for()` wykonującej swój blok instrukcji 10-krotnie mógłby wyglądać następująco:

```
for(int i=1; i<=10; i++)
{
    // blok instrukcji powtarzany 10 razy
}
```

4. Ćwiczenie 1. Do wykonania wspólnie z nauczycielem.

Napisz program, który na wyświetlaczu LED będzie odliczał cyfry od 0 do 9. Następnie będzie powtarzał odliczanie ponownie od 0 do 9. I takie odliczanie będzie kontynuowane przez cały czas działania programu.

Odstęp czasowy między kolejnymi cyframi powinien wynosić ok 1 sekundy.

Kod źródłowy rozwiązania ćwiczenia:

```
#include "Adafruit_MCP23008.h"

Adafruit_MCP23008 seg7;
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
```

```
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
  // konfiguracja pinów ekspandera
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
}

void loop() {
 for(int i=0; i<10;i++)
 {
    seg7.writeGPIO(digit[i]);
    delay(1000);
 }
}
```

5. Ćwiczenie 2. Samodzielnie do wykonania przez uczniów.

Napisz program, w którym zaprogramujesz ruch symbolu „*" (gwiazdki) na wyświetlaczu LCD.

Symbol „*" powinien sprawiać wrażenie, że przemieszcza się po obwodzie wyświetlacza. Tzn., początkowo gwiazdka znajduje się w lewym górnym rogu, a następnie co 0,5 sekundy przemieszcza się o jedną pozycję w prawo. Jeśli „dojdzie" do prawej krawędzi wyświetlacza LCD, to przemieszcza się do drugiego rzędu, ostatniej kolumny. Następnie „biegnie" ponownie do lewego boku wyświetlacza.

Takie okrążenia powinny się powtarzać, aż do wyłączenia urządzenia.

Kod źródłowy rozwiązania ćwiczenia:

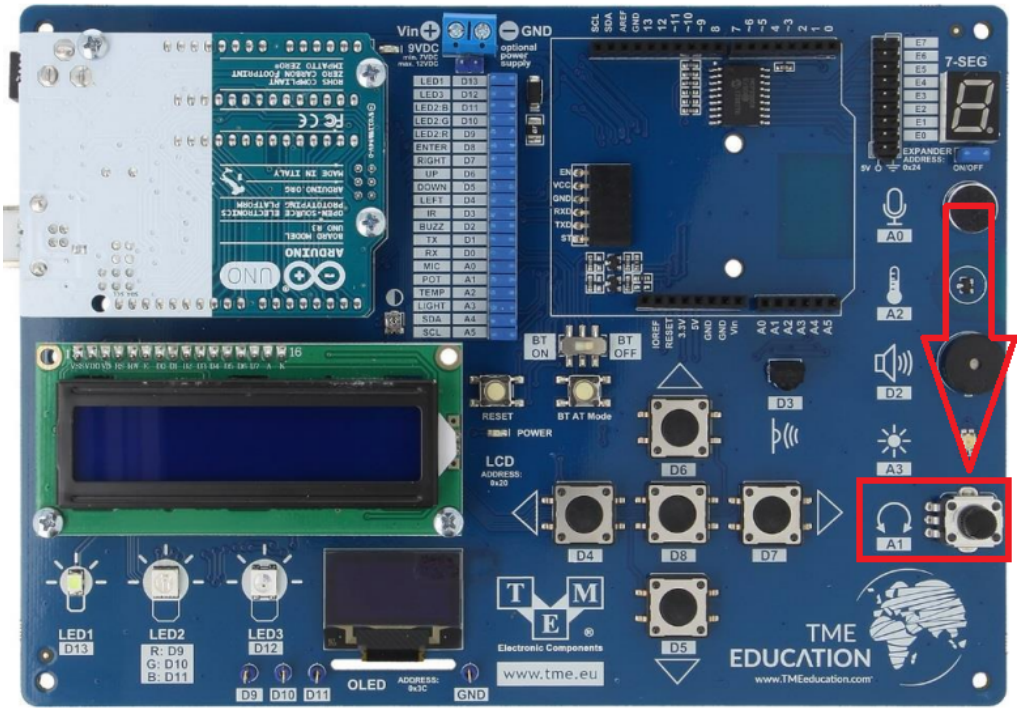
```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);
```

```
void setup() {
// konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {

  for(int i=0;i<16;i++)
  {
    lcd.clear();
    lcd.setCursor(i, 0);
    lcd.print("*");
    delay(500);
  }
  for(int j=15;j>=0;j--)
  {
    lcd.clear();
    lcd.setCursor(j, 1);
    lcd.print("*");
    delay(500);
  }
}
```

6. Omówienie przez nauczyciela działania i obsługi potencjometru.

Potencjometr jest kolejnym podzespołem, którym możemy programować komunikację użytkownika z układem Arduino UNO. Potencjometr to w pewnym sensie regulator wartości. Dosłownie jest to regulator rezystancji. Podłączenie go w zestawie edukacyjnym TME-EDU-ARD-2 umożliwia regulowanie nim napięcia dochodzącego do jednego z wejść układu Arduino UNO. Pin, do którego podłączony jest potencjometr, to pin A1. Jest to specjalny pin umożliwiający płynny odczyt wartości napięcia w zakresie od 0V do 5V. Dotąd poznane wejścia umożliwiały jedynie odczyt czy napięcie jest 0V (`LOW`), czy jest wysokie czyli 5V (`HIGH`). Tu na wejściu A1, tzw. Wejściu analogowym możemy odczytać już pośrednie wartości napięć.

Napięcie na wejściu analogowym w układzie Arduino UNO wyrażane jest w 10-bitowej skali. To znaczy, że przy odczycie otrzymujemy wartości od 0 do 1023. Aby je zinterpretować, należy przyjąć, że wartość odczytana 0 oznacza 0V, a wartość 1023 oznacza 5V na wejściu analogowym. Wartości spomiędzy tych wartości skrajnych oznaczają proporcjonalnie pośrednie wartości napięć od 0 do 5V.



7. Ćwiczenie 3. Do omówienia z nauczycielem.

Zmodyfikuj program z ćwiczenia 2 tak, by gwiazdka „*" przesuwała się po ekranie LCD z prędkością w zakresie od 0 milisekund do 1023 milisekund.

Do regulacji prędkości wykorzystaj potencjometr i wejście analogowe A1.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
// konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  for(int i=0;i<16;i++)
  {
    lcd.clear();
    lcd.setCursor(i, 0);
    lcd.print("*");
    // odczytanie wartości wejścia analogowego
    delay(analogRead(A1));
  }
```

```
  for(int j=15;j>=0;j--)
  {
    lcd.clear();
    lcd.setCursor(j, 1);
    lcd.print("*");
    // odczytanie wartości wejścia analogowego
    delay(analogRead(A1));
  }
}
```

W funkcji `delay()` jako parametr podano instrukcję odczytującą dane z pinu wejściowego A1 `delay(analogRead(A1));`.

Jeżeli chcemy skorzystać z wejścia analogowego, nie musimy dokonywać nowych konfiguracji w funkcji `setup()`.

8. Ćwiczenie 4 Do samodzielnej realizacji przez uczniów.

Napisz program, który dla skrajnego lewego wychylenia potencjometru na wyświetlaczu LED wskaże „0”. Dla skrajnego prawego wychylenia potencjometru wskaże wartość „9”. Dla pośrednich wychyleń potencjometru na wyświetlaczu powinny pojawić się proporcjonalnie policzone wartości z przedziału od 0 do 9.

Kod źródłowy rozwiązania ćwiczenia:

```
#include "Adafruit_MCP23008.h"

Adafruit_MCP23008 seg7;
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
  // konfiguracja pinów ekspandera
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
```

```
  seg7.pinMode(6, OUTPUT);
  seg7.pinMode(7, OUTPUT);
}

int i;
void loop() {
    i=analogRead(A1)/(1023/9);
    seg7.writeGPIO(digit[i]);
    delay(200);
}
```

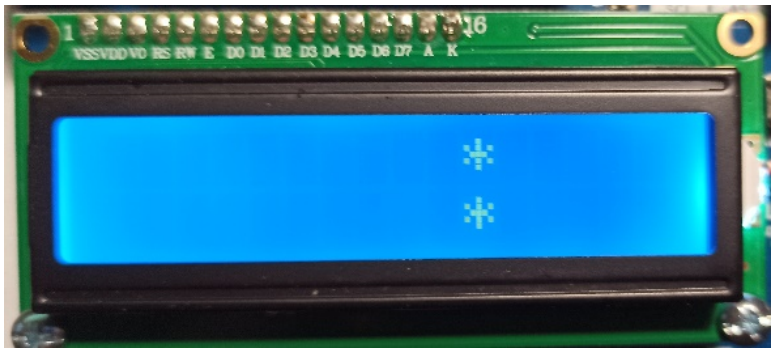
W programie do obliczenia cyfry jaką powinniśmy wyświetlić na wyświetlaczu LCD wartość odczytaną z wejścia analogowego podzielono przez dziewiątą cześć maksymalnej wartości jaka może pojawić się na wejściu analogowym. Wartość 9 wynika z faktu, iż wyświetlając dane na wyświetlaczu LED maksymalną wartość mamy 9.

9. Ćwiczenie 5 Do samodzielnej realizacji przez uczniów.

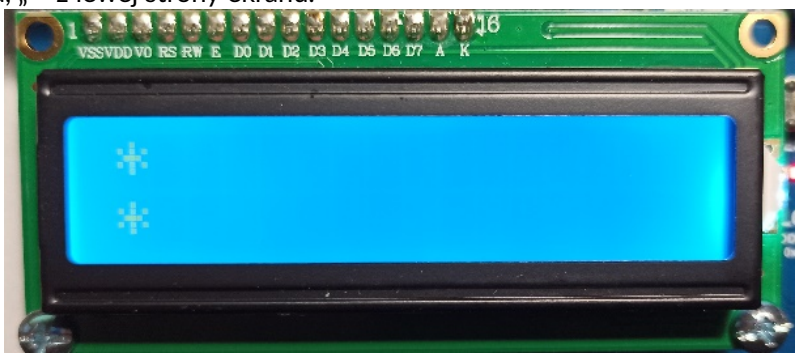
Napisz program, który za pomocą znaków „*” umieszczonych w po jednej sztuce w obu wierszach wyświetlacza LCD, wskaże pozycję potencjometru.

Dla skrajnego prawego wychylenia potencjometru „*” powinny znajdować się z prawej strony.

Dla wartości pośredniej:



I dla skrajnie lewej pozycji potencjometru, „*” z lewej strony ekranu.



Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
// konfiguracja LCD
  lcd.begin(16, 2);
}

int i;
void loop() {
    i=analogRead(A1)/(1023/15);
    lcd.clear();
    lcd.setCursor(i, 0);
    lcd.print("*");
    lcd.setCursor(i, 1);
    lcd.print("*");
    delay(200);
}
```

W programie do obliczenia pozycji „*” wartość odczytaną z wejścia analogowego podzielono przez piętnastą cześć maksymalnej wartości jaka może pojawić się na wejściu analogowym. Wartość 15 wynika z faktu, iż wyświetlając dane na wyświetlaczu LCD pozycję pisanego znaku określa się w przedziale od 0 do 15.

10. Przeprowadzenie quizu podsumowującego lekcję.

1. Pętla `for()`
   a) Jest identyczna jak pętla `while()`
   b) Ma jeden parametr
   c) Stosowana jest najczęściej do konkretnej ilości powtórzeń instrukcji
   d) Zawsze stosowana jest na końcu programu.
2. Poprawny nagłówek funkcji `for` to:

a) `for(int i=1;i<10;i++)`
b) `for (i<10;int i=1; i++)`
c) `for (i<10)`
d) `for(i++;i<10;int i=1)`

3. Pętla `for(int j=0;j<99;j++)` wykona się:
   a) 98 razy
   b) 99 razy
   c) 100 razy
   d) 101 razy
4. Funkcja `analogRead(A1)`
   a) Służy do odczytu stanu przycisków
   b) Maksymalnie osiąga wartość 5
   c) Służy do odczytu napięcia
   d) Jest równoznaczna z instrukcją `digitalRead(A1)`

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który na wyświetlaczu LED będzie odliczał cyfry od 0 do 9. Następnie będzie powtarzał odliczanie ponownie od 0 do 9. I takie odliczanie będzie kontynuowane przez cały czas działania programu.

Odstęp czasowy między kolejnymi cyframi powinien wynosić ok 1 sekundy.

Kod źródłowy rozwiązania ćwiczenia:

```
#include "Adafruit_MCP23008.h"

Adafruit_MCP23008 seg7;
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
  // konfiguracja pinów ekspandera
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
}

void loop() {
 for(int i=0; i<10;i++)
 {
```

```
        seg7.writeGPIO(digit[i]);
        delay(1000);
    }
}
```

Ćwiczenie 2

Napisz program, w którym zaprogramujesz ruch symbolu „*” (gwiazdki) na wyświetlaczu LCD.

Symbol „*” powinien sprawiać wrażenie, że przemieszcza się po obwodzie wyświetlacza. Tzn., początkowo gwiazdka znajduje się w lewym górnym rogu, a następnie co 0,5 sekundy przemieszcza się o jedną pozycję w prawo. Jeśli „dojdzie” do prawej krawędzi wyświetlacza LCD, to przemieszcza się do drugiego rzędu, ostatniej kolumny. Następnie „biegnie” ponownie do lewego boku wyświetlacza.

Takie okrążenia powinny się powtarzać, aż do wyłączenia urządzenia.

Ćwiczenie 3

Zmodyfikuj program z ćwiczenia 2 tak, by gwiazdka „*” przesuwała się po ekranie LCD z prędkością w zakresie od 0 milisekund do 1023 milisekund.

Do regulacji prędkości wykorzystaj potencjometr i wejście analogowe A1.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

void setup() {
// konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  for(int i=0;i<16;i++)
  {
    lcd.clear();
    lcd.setCursor(i, 0);
    lcd.print("*");
    // odczytanie wartości wejścia analogowego
    delay(analogRead(A1));
  }
  for(int j=15;j>=0;j--)
  {
    lcd.clear();
```

```
        lcd.setCursor(j, 1);
        lcd.print("*");
        // odczytanie wartości wejścia analogowego
        delay(analogRead(A1));
      }
    }
```

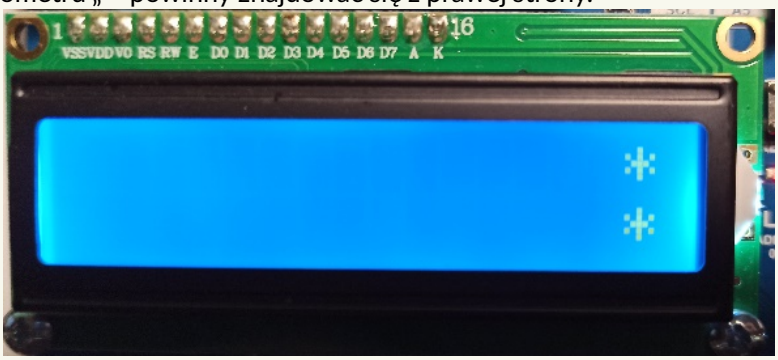
Ćwiczenie 4

Napisz program, który dla skrajnego lewego wychylenia potencjometru na wyświetlaczu LED wskaże „0". Dla skrajnego prawego wychylenia potencjometru wskaże wartość „9". Dla pośrednich wychyleń potencjometru na wyświetlaczu powinny pojawić się proporcjonalnie policzone wartości z przedziału od 0 do 9.

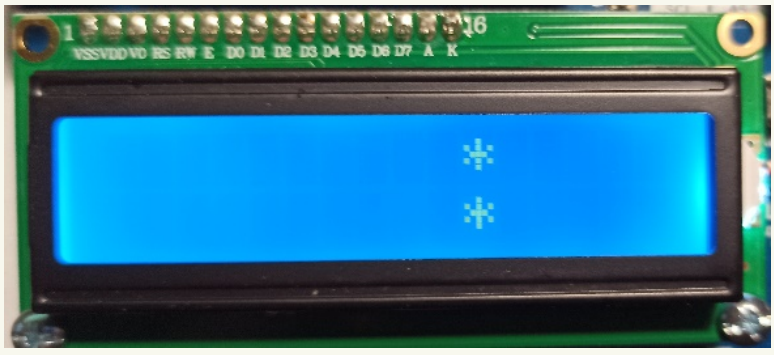
Ćwiczenie 5

Napisz program, który za pomocą znaków „*" umieszczonych w po jednej sztuce w obu wierszach wyświetlacza LCD, wskaże pozycję potencjometru.

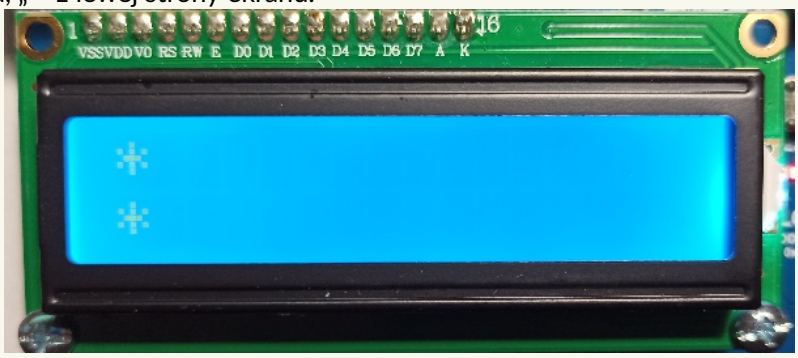
Dla skrajnego prawego wychylenia potencjometru „*" powinny znajdować się z prawej strony.



Dla wartości pośredniej:



I dla skrajnie lewej pozycji potencjometru, „*” z lewej strony ekranu.



Quiz

1. Pętla `for()`
   a) Jest identyczna jak pętla `while()`
   b) Ma jeden parametr
   c) Stosowana jest najczęściej do konkretnej ilości powtórzeń instrukcji
   d) Zawsze stosowana jest na końcu programu.
2. Poprawny nagłówek funkcji `for` to:
   a) `for(int i=1;i<10;i++)`
   b) `for (i<10;int i=1; i++)`
   c) `for (i<10)`
   d) `for(i++;i<10;int i=1)`
3. Pętla `for(int j=0;j<99;j++)` wykona się:
   a) 98 razy
   b) 99 razy
   c) 100 razy
   d) 101 razy
4. Funkcja `analogRead(A1)`
   a) Służy do odczytu stanu przycisków
   b) Maksymalnie osiąga wartość 5
   c) Służy do odczytu napięcia
   d) Jest równoznaczna z instrukcją `digitalRead(A1)`

Lekcja 8

Funkcje własne w języku C++. Wprowadzamy dane liczbowe do dalszych obliczeń w zestawie edukacyjnym TME-EDU-ARD-2.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych w języku C++. Utrwalenie wiedzy nt. stosowania pętli `while()` i pętli `for()`. Utrwalenie umiejętności odczytu danych z potencjometru, wyświetlacza LED i LCD oraz przycisków monostabilnych w zestawie edukacyjnym TME-EDU-ARD-2. Zapoznanie z funkcjami własnymi w języku C++.

Efekty kształcenia

- Uczeń potrafi stosować zmienne.
- Uczeń wie jak i kiedy stosuje się pętle `while()`.
- Uczeń stosuje pętle `while()` w swoich programach z Arduino UNO.
- Uczeń zna i stosuje funkcje własne w programach języka C++.
- Uczeń stosuje w swoich programach potencjometr, wyświetlacz LED i LCD oraz przyciski monostabilne do wprowadzania i wyświetlania danych.

Wstęp

W tej lekcji utrwalimy umiejętności posługiwania się poznanymi już wcześniej podzespołami jakimi jest wyświetlacz LCD i wyświetlacz LED, przyciski monostabilne, potencjometr i buzzer. Ponadto nauczymy się wprowadzać ww. podzespołami kilka danych liczbowych, które następnie zostaną poddane dalszym obliczeniom. Te wszystkie przykłady będą oparte na ćwiczeniach w których zbudujemy funkcje własne w języku C++.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

1. Jak programować poznane na poprzednich lekcjach podzespoły, tj. Przyciski monostabilne, dioda LED D1, buzzer, wyświetlacz LCD, wyświetlacz LED?
2. Powtórzenie przez nauczyciela, czym jest pętla `while()`. Kiedy i jak ją stosujemy?

3. Powtórzenie poznanych typów zmiennych liczb całkowitych, liczb rzeczywistych, łańcuchów znaków i zmiennych logicznych.

3. Ćwiczenie 1 Do samodzielnej realizacji przez uczniów.

Napisz program, który pozwoli użytkownikowi wprowadzić dwie różne liczby całkowite `a` i `b`. Liczby powinny zawierać się w przedziale od 0 do 1000. Liczby powinny zostać wyświetlone w pierwszej linii wyświetlacza LCD. Pierwsza liczba od lewej strony, druga liczba od prawej strony. Zatwierdzenie wprowadzania poszczególnych liczb powinno odbywać się za pomocą przycisku `SW_CENTER`.

Przykładowy efekt działania programu:

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_CENTER 8

int a=0;
int b=0;

// zmienna pomocnicza przechowująca informacje o liczbie,
// którą w danym momencie wprowadza użytkownik:
// jeżeli 0, to użytkownik wprowadza zmienną a
// jeżeli 1, to użytkownik wprowadza zmienną b
bool ktora_liczba=0;

void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
```

```
    if(ktora_liczba==0)
      a=analogRead(A1)/1.023;
    else
      b=analogRead(A1)/1.023;

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("a=");
    lcd.setCursor(2, 0);
    lcd.print(a);

    lcd.setCursor(10, 0);
    lcd.print("b=");
    lcd.setCursor(12, 0);
    lcd.print(b);

    if (digitalRead(SW_CENTER) == HIGH)
    {
      // negacja(odwrócenie wartości przechowywanej w zmiennej
      ktora_liczba=!ktora_liczba;
    }
    delay(200);
  }
```

W programie utworzono zmienną `ktora_liczba`. Informuje ona, dla której z liczb program ma w danym momencie przypisać wartość z potencjometru. Zmienna ta służy kontroli, która z liczb, `a` czy `b`, jest aktualnie wprowadzana. Jeśli zmienna `ktora_liczba` ma wartość 0, to wprowadzamy wartość dla zmiennej `a`. Kiedy ma wartość 1, to wprowadzamy wartość dla zmiennej `b`. Działanie `analogRead(A1)/1.023;` służy ograniczeniu wartości odczytywanej do max 1000 (Zgodnie z założeniem w treści ćwiczenia). Instrukcja `ktora_liczba=!ktora_liczba;` skutkuje zmianą wartości zmiennej logicznej `która_liczba` na przeciwną, czyli zanegowaniem jej.

4. Ćwiczenie 2 Do realizacji wraz z nauczycielem.

Napisz program, który tak jak w ćwiczeniu 1 pozwoli użytkownikowi wprowadzić dwie różne liczby całkowite. Zmodyfikuj ten program tak, by pod wprowadzaną liczbą lub jej symbolem mrugał kursor. W chwili zatwierdzenia przyciskiem `SW_CENTER` powinien z buzzera wydobywać się krótki dźwięk.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_CENTER 8
```

```
// definiujemy stałą buzzer
#define BUZZER 2

int a=0;
int b=0;

// zmienna pomocnicza wskazująca, którą z liczb w danym momencie wprowadza użytkownik.
// jeżeli użytkownik wprowadza zmienną a, to 0
// jeżeli użytkownik wprowadza zmienną b, to 1
bool ktora_liczba=0;

int wait=0;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  if(ktora_liczba==0)
    a=analogRead(A1)/1.023;
  else
    b=analogRead(A1)/1.023;

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("a=");
  lcd.setCursor(2, 0);
  lcd.print(a);

  lcd.setCursor(10, 0);
  lcd.print("b=");
  lcd.setCursor(12, 0);
  lcd.print(b);

  if (digitalRead(SW_CENTER) == HIGH)
  {
    // negacja(odwrócenie wartości przechowywanej w zmiennej
    ktora_liczba=!ktora_liczba;
    tone(BUZZER, 2000);
    delay(200);
    noTone(BUZZER);
  }
  if(ktora_liczba==0)
    lcd.setCursor(0, 0);
  else
    lcd.setCursor(10, 0);
  if(wait == 0)
    {
```

```
        // funkcja włączająca kursor
        lcd.cursor();
        wait = 1;
      }

      else
      {
        // funkcja wyłączająca kursor
        lcd.noCursor();
        wait = 0;
      }
    delay(500);
}
```

W powyższym ćwiczeniu dodano funckcję `lcd.cursor();` oraz `lcd.noCursor();`. Funkcja `lcd.cursor();` włącza miganie kursora we wskazanej przez `lcd.setCursor();` pozycji. Funkcja `lcd.noCursor();` wyłącza miganie kursora i kursor znika.

5. Omówienie przez nauczyciela funkcji własnych w języku C++

Funkcje własne w języku C++ służą oznaczeniu pewnych instrukcji (bloku instrukcji) unikatową nazwą. Dzięki takiemu oznaczeniu będziemy mogli w każdym miejscu programu przytoczyć nazwę naszej funkcji i tym samym wywołać realizację bloku instrukcji w niej się znajdujących. Takie wywołanie może następować wielokrotnie. Czyli raz napisany kod bloku instrukcji składający się z wielu wierszy może być później wywoływany przez podanie jedynie nazwy naszej własnej funkcji.

Istnieje kilka typów funkcji:

Funkcja, która nie zwraca żadnej wartości i nie przyjmuje argumentów:

```
void nazwa_funkcji()
{
    // instrukcje do realizacji
}
```

Funkcja, która nie zwraca żadnej wartości i przyjmuje argumenty:

```
void nazwa_funkcji(lista_argumentów)
{
    // instrukcje do realizacji
}
```

Funkcja, która zwraca wartość i nie przyjmuje argumentów:

```
Typ_zwracanej_zmiennej nazwa_funkcji()
{
    // instrukcje do realizacji
        return zwracana_wartosc;
}
```

Funkcja, która zwraca wartość i przyjmuje argumenty:

```
Typ_zwracanej_zmiennej nazwa_funkcji(lista_argumentów)
{
    // instrukcje do realizacji
        return zwracana_wartosc;
}
```

Wywołanie takiej funkcji odbywa się przez użycie jej nazwy i wprowadzenia ewentualnej listy argumentów, np.:

```
nazwa_funkcji(lista_argumentów);
```

6. Ćwiczenie 3 Do realizacji wraz z nauczycielem.

Zmodyfikuj swój program z ćwiczenia 2 tak, by w osobnych funkcjach własnych znalazły się następujące funkcjonalności:

- Odczyt wartości `a` i `b`
- Wyświetlanie wartości liczb `a` i `b`
- Mruganie kursora
- Zatwierdzenie liczb

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_CENTER 8
// definiujemy stałą buzzer
#define BUZZER 2

int a=0;
int b=0;

// zmienna pomocnicza wskazująca, którą z liczb w danym momencie wprowadza użytkownik.
// jeżeli użytkownik wprowadza zmienną a, to 0
// jeżeli użytkownik wprowadza zmienną b, to 1
bool ktora_liczba=0;

int wait=0;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

// funkcja nie przyjmująca argumentów i nie zwracająca wartości
// funkcja służąca do odcztu liczb z potencjometru
```

```
void odczyt()
{
  if(ktora_liczba==0)
    a=analogRead(A1)/1.023;
  else
    b=analogRead(A1)/1.023;
}

// funkcja wyświetlająca dane o zmiennych a i b na wyświetlaczu
void wyswietl(int a, int b)
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("a=");
  lcd.setCursor(2, 0);
  lcd.print(a);

  lcd.setCursor(10, 0);
  lcd.print("b=");
  lcd.setCursor(12, 0);
  lcd.print(b);
}

// funkcja sprawdzająca, czy liczba jest zatwierdzona
void czy_zatwierdz()
{
  if (digitalRead(SW_CENTER) == HIGH)
  {
    // negacja(odwrócenie wartości przechowywanej w zmiennej
    ktora_liczba=!ktora_liczba;
    tone(BUZZER, 2000);
    delay(200);
    noTone(BUZZER);
  }
}

// funkcja odpowiedzialna za pojedyncze mignięcie kursora
void miganie()
{
  if(ktora_liczba==0)
    lcd.setCursor(0, 0);
  else
    lcd.setCursor(10, 0);
  if(wait == 0)
    {
      // funkcja włączająca kursor
      lcd.cursor();
```

```
        wait = 1;
      }
      else
      {
        // funkcja wyłączająca kursor
        lcd.noCursor();
        wait = 0;
      }
  }

  void loop() {
    odczyt();
    wyswietl(a,b);
    czy_zatwierdz();
    miganie();
    delay(500);
  }
```

W rozwiązaniu ćwiczenia funkcjonalności przyjętych rozwiązań zostały przypisane do następujących funkcji własnych użytkownika:

- `odczyt();` - odczyt danych z wejścia naalogowego (potencjometru)
- `wyswietl(a,b);` - wyświetlenie danych o wartościach a i b na wyświetlaczu LCD
- `czy_zatwierdz();` - analiza czy naciśnięto przycisk zatwierdzenia wprowadzonej danej
- `miganie();` - funkcja odpowiedzialna za miganie kursora przy wprowadzanej wartości.

Wszystkie funkcje są wywołane w funkcji `void loop()`.

7. Ćwiczenie 4 Dla zaawansowanych

Napisz program, który po zatwierdzeniu drugiej liczby z ćwiczenia 3 (zmienna `b`) da możliwość wyboru działania arytmetycznego, które wykonane będzie na wprowadzonych wartościach.

Działanie arytmetyczne powinno być wybierane za pomocą przycisków monostabilnych `SW_LEFT` i `SW_RIGHT`. Zatwierdzenie działania i wyświetlenie wyniku `SW_CENTER`.

Przykładowe wyświetlenie wyboru działania arytmetycznego:

Wynik powinien pojawić się w drugim wierszu wyświetlacza LCD.

Przykładowe wyświetlenie wyniku (SUMA):

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_CENTER 8
#define SW_RIGHT 7
#define SW_LEFT 4

// definiujemy stałą buzzer
#define BUZZER 2

int a=0;
int b=0;

// zmienna pomocnicza wskazująca, którą z liczb w danym momencie wprowadza użytkownik.
// jeżeli użytkownik wprowadza zmienną a, to 0
// jeżeli użytkownik wprowadza zmienną b, to 1
bool ktora_liczba=0;

int dzialanie=1;
int wait=0;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

// funkcja nie przyjmująca argumentrów i nie zwracająca wartści
// funkcja służąca do odcztyu liczb z potencjometru
void odczyt()
{
  if(ktora_liczba==0)
```

```
    a=analogRead(A1)/1.023;
  else
    b=analogRead(A1)/1.023;
}

// funkcja wyświetlająca dane o zmiennych a i b na wyświetlaczu
void wyswietl(int a, int b)
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("a=");
  lcd.setCursor(2, 0);
  lcd.print(a);

  lcd.setCursor(10, 0);
  lcd.print("b=");
  lcd.setCursor(12, 0);
  lcd.print(b);
}
void dzialania()
{
  if(dzialanie==1)
  {
    lcd.setCursor(0, 1);
    lcd.print("<     SUMA     >");
  }
  if(dzialanie==2)
  {
    lcd.setCursor(0, 1);
    lcd.print("<   ROZNICA    >");
  }
  if(dzialanie==3)
  {
    lcd.setCursor(0, 1);
    lcd.print("<   ILOCZYN    >");
  }
  if(dzialanie==4)
  {
    lcd.setCursor(0, 1);
    lcd.print("<    ILORAZ    >");  }
  if (digitalRead(SW_RIGHT) == HIGH && dzialanie<4)
    dzialanie++;
  if (digitalRead(SW_LEFT) == HIGH && dzialanie >1)
    dzialanie--;
  delay(500);
```

```
}
// funkcja sprawdzająca czy liczba jest zatwierdzona
void czy_zatwierdz()
{
  if (digitalRead(SW_CENTER) == HIGH)
  {
    if(ktora_liczba==0)
    {
      // negacja(odwrócenie wartości przechowywanej w zmiennej
      ktora_liczba=!ktora_liczba;
    }
    else
    {
      delay(500);
      while(digitalRead(SW_CENTER) == LOW)
      {
        dzialania();
      }
      wyswietl(a,b);
      if(dzialanie==1)
      {
        lcd.setCursor(0, 1);
        lcd.print(suma(a,b));
      }
      if(dzialanie==2)
      {
        lcd.setCursor(0, 1);
        lcd.print(roznica(a,b));
      }
      if(dzialanie==3)
      {
        lcd.setCursor(0, 1);
        lcd.print(iloczyn(a,b));
      }
      if(dzialanie==4)
      {
        lcd.setCursor(0, 1);
        lcd.print(iloraz(a,b));  }
        while(1){}// pętla nieskończona
    }
    tone(BUZZER, 2000);
    delay(200);
    noTone(BUZZER);
  }
}
```

```
// funkcja odpowiedzialna za pojedyncze mignięcie kursora
void miganie()
{
  if(ktora_liczba==0)
    lcd.setCursor(0, 0);
  else
    lcd.setCursor(10, 0);
  if(wait == 0)
    {
      // funkcja włączająca kursor
      lcd.cursor();
      wait = 1;
    }
    else
    {
      // funkcja wyłączająca kursor
      lcd.noCursor();
      wait = 0;
    }
}

int suma(int a, int b)
{
  return a+b;
}
int roznica(int a, int b)
{
  return a-b;
}
int iloczyn(int a, int b)
{
  return a*b;
}
float iloraz(int a, int b)
{
  if(b!=0)
    return float(a)/float(b);
}

void loop() {
  odczyt();
  wyswietl(a,b);
  czy_zatwierdz();
  miganie();
  delay(500);
}
```

W ćwiczeniu wprowadzono funkcję `void dzialania()`, która odpowiada za wyświetlenie menu dostępnych działań arytmetycznych. Ponadto w funkcji `void czy_zatwierdz()` dodano obsługę funkcji arytmetycznej.

8. Przeprowadzenie quizu podsumowującego lekcję.

1. Funkcja `lcd.setCursor(1, 0);` ustawi kursor na wyświetlaczu LCD w pozycji:
   a) Drugiego znaku drugiego wiersza
   b) Drugiego znaku pierwszego wiersza
   c) Pierwszego znaku drugiego wiersza
   d) Drugiego znaku pierwszego wiersza
2. Wejście analogowe A1 mierzy napięcie na wejściu w skali:
   a) Od 0 do 1023
   b) Od 0 do 1000
   c) Od 1 do 1024
   d) Od 0 do 1024
3. Symbol wykrzyknika „!” przed zmienną logiczną oznacza że:
   a) Do zmiennej będzie dodana wartość 1
   b) Od zmiennej zostanie odjęta wartość 1
   c) Zmienna będzie zanegowana
   d) Zmienna w razie niepoprawnego zapisu będzie poprawiona.
4. Funkcja `lcd.cursor();` :
   a) Ustawia kursor na wyświetlaczu w celu wpisania znaków
   b) Włącza widoczność kursora
   c) Wyłącza widoczność kursora
   d) Włącza wyświetlacz LCD
5. Funkcja

```
float iloraz(int a, int b)
{
  if(b!=0)
    return float(a)/float(b);
}
```

   jest funkcją:
   a) Zwracającą wartość całkowitą i przyjmującą parametry całkowite
   b) Zwracającą wartość rzeczywistą i przyjmującą parametry całkowite
   c) Zwracającą wartość całkowitą i przyjmującą parametry rzeczywiste
   d) Niezwracającą wartości i przyjmującą parametry całkowite

Prawidłowe odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który pozwoli użytkownikowi wprowadzić dwie różne liczby całkowite `a` i `b`. Liczby powinny zawierać się w przedziale od 0 do 1000. Liczby powinny zostać wyświetlone w pierwszej linii wyświetlacza LCD. Pierwsza liczba od lewej strony, druga liczba od prawej strony. Zatwierdzenie wprowadzania poszczególnych liczb powinno odbywać się za pomocą przycisku `SW_CENTER`.

Przykładowy efekt działania programu:

Ćwiczenie 2

Napisz program, który tak jak w ćwiczeniu 1 pozwoli użytkownikowi wprowadzić dwie różne liczby całkowite. Zmodyfikuj ten program tak, by pod wprowadzaną liczbą lub jej symbolem mrugał kursor. W chwili zatwierdzenia przyciskiem `SW_CENTER` powinien z buzzera wydobywać się krótki dźwięk.

Kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_CENTER 8

// definiujemy stałą buzzer
#define BUZZER 2
```

```
int a=0;
int b=0;

// zmienna pomocnicza wskazująca, którą z liczb w danym momencie wprowadza użytkownik.
// jeżeli użytkownik wprowadza zmienną a, to 0
// jeżeli użytkownik wprowadza zmienną b, to 1
bool ktora_liczba=0;

int wait=0;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  if(ktora_liczba==0)
    a=analogRead(A1)/1.023;
  else
    b=analogRead(A1)/1.023;

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("a=");
  lcd.setCursor(2, 0);
  lcd.print(a);

  lcd.setCursor(10, 0);
  lcd.print("b=");
  lcd.setCursor(12, 0);
  lcd.print(b);

  if (digitalRead(SW_CENTER) == HIGH)
  {
    // negacja(odwrócenie wartości przechowywanej w zmiennej
    ktora_liczba=!ktora_liczba;
    tone(BUZZER, 2000);
    delay(200);
    noTone(BUZZER);
  }
  if(ktora_liczba==0)
    lcd.setCursor(0, 0);
  else
    lcd.setCursor(10, 0);
  if(wait == 0)
    {
      // funkcja włączająca kursor
      lcd.cursor();
      wait = 1;
```

```
        }
        else
        {
          // funkcja wyłączająca kursor
          lcd.noCursor();
          wait = 0;
        }
      delay(500);
    }
```

Ćwiczenie 3

Zmodyfikuj swój program z ćwiczenia 2 tak, by w osobnych funkcjach własnych znalazły się następujące funkcjonalności:

- Odczyt wartości `a` i `b`
- Wyświetlanie wartości liczb `a` i `b`
- Mruganie kursora
- Zatwierdzenie liczb

Ćwiczenie 4

Napisz program, który po zatwierdzeniu drugiej liczby z ćwiczenia 3 (zmienna `b`) da możliwość wyboru działania arytmetycznego, które wykonane będzie na wprowadzonych wartościach.

Działanie arytmetyczne powinno być wybierane za pomocą przycisków monostabilnych `SW_LEFT` i `SW_RIGHT`. Zatwierdzenie działania i wyświetlenie wyniku `SW_CENTER`.

Przykładowe wyświetlenie wyboru działania arytmetycznego:

Wynik powinien pojawić się w drugim wierszu wyświetlacza LCD.

Przykładowe wyświetlenie wyniku (SUMA):


1
16
VSSVDD VO RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A K
a=384
b=83
467_

Quiz

1. Funkcja `lcd.setCursor(1, 0);` ustawi kursor na wyświetlaczu LCD w pozycji:
   a) Drugiego znaku drugiego wiersza
   b) Drugiego znaku pierwszego wiersza
   c) Pierwszego znaku drugiego wiersza
   d) Drugiego znaku pierwszego wiersza
2. Wejście analogowe A1 mierzy napięcie na wejściu w skali:
   a) Od 0 do 1023
   b) Od 0 do 1000
   c) Od 1 do 1024
   d) Od 0 do 1024
3. Symbol wykrzyknika „!” przed zmienną logiczną oznacza że:
   a) Do zmiennej będzie dodana wartość 1
   b) Od zmiennej zostanie odjęta wartość 1
   c) Zmienna będzie zanegowana
   d) Zmienna w razie niepoprawnego zapisu będzie poprawiona.
4. Funkcja `lcd.cursor();` :
   a) Ustawia kursor na wyświetlaczu w celu wpisania znaków
   b) Włącza widoczność kursora
   c) Wyłącza widoczność kursora
   d) Włącza wyświetlacz LCD
5. Funkcja
   jest funkcją:
   a) Zwracającą wartość całkowitą i przyjmującą parametry całkowite
   b) Zwracającą wartość rzeczywistą i przyjmującą parametry całkowite
   c) Zwracającą wartość całkowitą i przyjmującą parametry rzeczywiste
   d) Niezwracającą wartości i przyjmującą parametry całkowite

Lekcja 9

Poznajemy modele barw. Generator kolorów RGB w zestawie edukacyjnym TME-EDU-ARD-2.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych i pętli w języku C++.
Utrwalenie umiejętności odczytu danych z potencjometru, wyświetlacza LCD oraz przycisków monostabilnych w zestawie edukacyjnym TME-EDU-ARD-2.
Zapoznanie z modelami barw stosowanymi w IT w szczególności z modelem RGB.
Zapoznanie z możliwościami sterowania diodą RGB w zestawie edukacyjnym TME-EDU-ARD-2.

Efekty kształcenia

- Uczeń potrafi stosować zmienne w języku C++.
- Uczeń wie jak i kiedy stosuje się pętle i instrukcje warunkowe w języku C++.
- Uczeń stosuje w swoich programach potencjometr, wyświetlacz LED i LCD oraz przyciski monostabilne do wprowadzania i wyświetlania danych.
- Uczeń wie czym są modele barw.
- Uczeń zna model barw RGB
- Uczeń potrafi przedstawić barwy modelu RGB na diodzie świecącej RGB.

Wstęp

W trakcie lekcji utrwalimy umiejętności kontroli nad wcześniej poznanymi podzespołami zestawu edukacyjnego, m.in. jakimi jest wyświetlacz LCD, przyciski monostabilne, potencjometr. Umiejętność programowania znanych z wcześniejszych lekcji pozesdpołów uzupełnimy poznaniem obsługi diody świecącej RGB. Nowopoznany podzespół będzie dla nas narzędziem poznania czym są modele barw i generowania kolorów w modelu barw RGB.

Przebieg lekcji

1. Przedstawienie celów lekcji.
2. Powtórzenie wiadomości z poprzednich lekcji,

1. Jak programować poznane na poprzednich lekcjach podzespoły, tj. Przyciski monostabilne, dioda LED D1, wyświetlacz LCD?
2. Powtórzenie przez nauczyciela, czym różnią sie pętle `while()`, `for()` i `do.. while()`.
3. Powtórzenie poznanych typów zmiennych.

3. Omówienie przez nauczyciela czym są modele barw.

Wszyskie widzialne barwy, możemy przedstawić w postaci matematycznych modeli. Modele te starają sie jak najlepiej odzwierciedlić ludzką percepcję i możliwości fizyczne oka.

Te modele matematyczne, które uznano za najważniejsze ujęto w normach. Modele barw stosowane są w różnych dziedzinach: fotograficznych, pożywczych, farbiarskich czy tekstylnych.

Wśród tych najważniejszych modeli barw możemy wyróżnić:

RGB,

CMYK

HSV

HSL

YUV

Model na którym skupimy swoją uwagę poświęcony jest addytywnemu mieszaniu barw światła i jest modelem RGB. Opisany jest on trzema ilościami udziału światła R(czerwonego), G(zielonego), B(niebieskiego). Dzięki dodawaniu trzech barw światła jesteśmy w stanie odwzorować niemal każdy kolor. Najpopularniejszy z wariantów to model RGB z zapisem 24- bitowym. W 24-bitowym zapisie na kążdą barwę przewidziano po 8-bitów, czyli dla każdego koloru możemy kontrolować jego intensywność w zakresie od 0 do 255.

Zsumowanie intensywności 0(min) dla każdej barwy skutkuje po zsumowaniu emisją barwy czarnej.

Zsumowanie intensywności 255 (max) dla każdej barwy skutkuje emisją światła białego.

4. Ćwiczenie 1 Do realizacji wraz z nauczycielem.

Napisz program, który wysteruje potencjometrem kolory czerwony, zielony i niebieski na diodzie RGB LED2. Sterowanie powinno polegać na tym, że skrajne lewe wychylenie powinno spowodować włączenie koloru czerwonego na diodzie RGB, pozycja środkowa potencjometru powinna włączyć kolor zielony, a pozycja skrajnie prawa kolor niebieski.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11

void setup() {
 // konfiguracja pinów diody RGB jako pinów wyjściowych
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
```

```
}

void loop() {
 if(analogRead(A1)<=(1024*0.33))
 {
    // włącz kolor czerwony
     digitalWrite(LED2RED, HIGH);
     digitalWrite(LED2GREEN, LOW);
     digitalWrite(LED2BLUE, LOW);
 }
  if(analogRead(A1)>(1024*0.33)&&analogRead(A1)<=(1024*0.67))
 {
    // włącz kolor zielony
     digitalWrite(LED2RED, LOW);
     digitalWrite(LED2GREEN, HIGH);
     digitalWrite(LED2BLUE, LOW);
 }
  if(analogRead(A1)>(1024*0.67))
 {
    // włącz kolor niebieski
     digitalWrite(LED2RED, LOW);
     digitalWrite(LED2GREEN, LOW);
     digitalWrite(LED2BLUE, HIGH);
 }
}
```

Dioda RGB znajdująca się w zestawie edukacyjnym wysterowywana jest tak, jakby sterować załączaniem trzech osobnych diod LED o kolorach czerwonym, zielonym i niebieskim. Również fizyczne podłączenie diody RGB do układu Arduino UNO zrealizowane jest na trzech oddzielnych pinach sterujących poszczególnymi barwami składowymi modelu RGB.

W programie na początku zdefiniowano trzy stałe programowe `LED2RED`, `LED2GREEN`, `LED2BLUE`, dla których przypisano numery wyprowadzeń z układu Arduino UNO. Następnie w funkcji `void setup()` skonfigurowano piny jako wyjściowe.

Zadaniem funkcji `void loop()` jest załączanie poszczególnych barw RGB w zależności od pozycji potencjometru. Analiza pozycji potencjometru odbywa się na podstawie odczytu wartości analogowego wejścia A1. Jeśli wynosi ona mniej niż jedna trzecia maksymalnej wartości na wejściu analogowym A1, załączany jest kolor czerwony diody RGB (przełączany jest na stan wysoki. Pozostałe diody zostają w tym czasie wyłączone. Dla pozostałych barw proces przebiega podobnie. Tzn., instrukcjami warunkowymi sprawdzane jest, czy odczyt z wejścia analogowego A1 znajduje się w ustalonym dla danego koloru przedziale – jeśli tak, to pozostałe diody są wyłączane, a odpowiedzialna za dany kolor jest załączana.

5. Ćwiczenie 2 Do samodzielnej realizacji przez uczniów.

Napisz program, który wysteruje przyciskami monostabilnymi włączenie bądź wyłączenie kolorów czerwonego, zielonego i niebieskiego na diodzie RGB LED2.

Pojedyncze naciśnięcie przycisku powinno zmieniać stan załączenia koloru, za którego sterowanie dany przycisk jest odpowiedzialny.

— przycisku `SW_LEFT` – koloru czerwonego,
— przycisku `SW_CENTER` – koloru zielonego,
— przycisku `SW_RIGHT` – koloru niebieskiego.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11
#define SW_CENTER 8
#define SW_RIGHT 7
#define SW_LEFT 4

bool sw_red=0;
bool sw_green=0;
bool sw_blue=0;

void setup() {
 // konfiguracja pinów diody RGB jako pinów wyjściowych
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
}

void loop() {

 if(digitalRead(SW_LEFT) == HIGH)
 {
    sw_red=!sw_red;
 }
 if(digitalRead(SW_CENTER) == HIGH)
 {
    sw_green=!sw_green;
 }

 if(digitalRead(SW_RIGHT) == HIGH)
 {
    sw_blue=!sw_blue;
 }

 if(sw_red==1)
 {
  // włącz kolor czerwony
  digitalWrite(LED2RED, HIGH);
 }

 else
```

```
{
 // wyłącz kolor czerwony
 digitalWrite(LED2RED, LOW);
}
if(sw_green==1)
{
 // włącz kolor zielony
 digitalWrite(LED2GREEN, HIGH);
}

else
{
 // wyłącz kolor zielony
 digitalWrite(LED2GREEN, LOW);
}
 if(sw_blue==1)
{
 // włącz kolor niebieski
 digitalWrite(LED2BLUE, HIGH);
}
else
{
 // wyłącz kolor niebieski
 digitalWrite(LED2BLUE, LOW);
}
delay(200);
}
```

W programie podobnie jak w ćwiczeniu 1 zdefiniowaliśmy stałe dla poszczególnych kolorów. Skonfigurowaliśmy odpowiednie piny układu Arduino UNO. Następnie utworzone zostały trzy zmienne logiczne `sw_red`, `sw_green`, `sw_blue` stanowiące flagi informujące o załączeniu bądź wyłączeniu poszczególnych barw. Flagi te będziemy zmieniać (negować) w sytuacji wykrycia naciśnięcia poszczególnych przycisków monostabilnych. Dla diody czerwonej stan flagi zmieni przycisk `SW_LEFT`. Dla diody zielonej stan flagi zmieni przycisk `SW_CENTER`. Dla diody niebieskiej stan flagi zmieni przycisk `SW_RIGHT`.

Po kontroli stanu przycisków monostabilnych i wartości poszczególnych flag załączane bądź wyłączane są poszczególne barwy światła diody RGB.

Funkcja opóźniająca `delay(200);` jest konieczna, gdyż po wciśnięciu przycisku monostabilnego ciągle kontrolowane jest jego naciśnięcie i zmieniane są stany flag. Aby umożliwić użytkownikowi puszczenie przycisku, dodano opóźnienie 200 ms.

6. Ćwiczenie 3 Do realizacji wraz z nauczycielem.

Napisz program, który w sposób płynny będzie przechodził między kolorami czerwonym, zielonym i niebieskim. Poszczególne kolory powinny rozświetlać się od 0 do maksymalnej swojej jasności, a następnie powinny się wygasić i przejść w kolejny kolor.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11

void setup() {
 // konfiguracja pinów
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
}

void loop() {
 for(int i=0; i<=255; i++)
 {
   // ustaw jasność diody czerwonej
   analogWrite(LED2RED, i);
   delay(10);
 }
 for(int i=255; i>=0; i--)
 {
   // ustaw jasność diody czerwonej
   analogWrite(LED2RED, i);
   delay(10);
 }

  for(int i=0; i<=255; i++)
 {
   // ustaw jasność diody zielonej
   analogWrite(LED2GREEN, i);
   delay(10);
 }
 for(int i=255; i>=0; i--)
 {
   // ustaw jasność diody zielonej
   analogWrite(LED2GREEN, i);
   delay(10);
 }

  for(int i=0; i<=255; i++)
 {
   // ustaw jasność diody niebieskiej
   analogWrite(LED2BLUE, i);
   delay(10);
 }
 for(int i=255; i>=0; i--)
 {
   // ustaw jasność diody niebieskiej
```

```
    analogWrite(LED2BLUE, i);
    delay(10);
  }
}
```

By wykonać sterowanie intensywnością poszczególnych barw diody RGB, a nie tylko załączeniem-wyłączeniem danej barwy, musimy posługiwać się poleceniem `analogWrite()`. Polecenie to zawiera dwa parametry. Pierwszy to numer pinu układu Arduino UNO, na którym sterowana jest intensywność danej barwy. Drugi parametr to wartość w zakresie od 0 do 255 określająca w ww. skali intensywność udziału danej barwy RGB.

Barwy mają zmieniać się od zerowej intensywności do maksymalnego natężenia. Następnie natężenie spada ponownie do 0. Do takiej czynności świetnie sprawdziły się pętle `for()`, które dla każdego koloru zmieniają wartość jego intensywności.

7. Ćwiczenie 4 Do samodzielnej realizacji przez uczniów.

Napisz program, który umożliwi sterowanie kolorem świecenia diody RGB. Sterowanie kolorem świecenie będzie odbywało się za pomocą potencjometru. O tym, który kolor jest w danej chwili sterowany, będzie decydowało naciśnięcie przycisków `SW_LEFT`, `SW_CENTER` i `SW_RIGHT`. `SW_LEFT` włącza sterowanie kolorem czerwonym. `SW_CENTER` włącza sterowanie kolorem zielonym, zaś `SW_RIGHT` włącza sterowanie kolorem niebieskim.

To, który kolor w danym momencie jest regulowany, powinien wskazywać kursor na wyświetlaczu LED, podkreślający odpowiednio litery R (czerwony), G (zielony) lub B (niebieski).

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11
#define SW_CENTER 8
#define SW_RIGHT 7
#define SW_LEFT 4

// definiujemy stałą buzzer
#define BUZZER 2

int ktory_kolor=1;
int valueRed=0;
int valueGreen=0;
int valueBlue=0;

void wyswietlNaLCD()
{
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("R=");
  lcd.setCursor(0, 1);
  lcd.print(valueRed);

  lcd.setCursor(6, 0);
  lcd.print("G=");
  lcd.setCursor(6, 1);
  lcd.print(valueGreen);

  lcd.setCursor(11, 0);
  lcd.print("B=");
  lcd.setCursor(11, 1);
  lcd.print(valueBlue);
}

void kursor()
{
  if(ktory_kolor==1)
    lcd.setCursor(0, 1);
  if(ktory_kolor==2)
    lcd.setCursor(6, 1);
  if(ktory_kolor==3)
    lcd.setCursor(11, 1);
lcd.cursor();
}

void setup() {
 // konfiguracja pinów
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);

 // konfiguracja LCD
  lcd.begin(16, 2);

}

void loop() {
   if (digitalRead(SW_LEFT) == HIGH)
  {
    ktory_kolor=1;
    tone(BUZZER, 1000);
    delay(200);
    noTone(BUZZER);
  }
 if (digitalRead(SW_CENTER) == HIGH)
  {
```

```
    ktory_kolor=2;
    tone(BUZZER, 2000);
    delay(200);
    noTone(BUZZER);
  }
 if (digitalRead(SW_RIGHT) == HIGH)
  {
    ktory_kolor=3;
    tone(BUZZER, 4000);
    delay(200);
    noTone(BUZZER);
  }
  if(ktory_kolor==1)
    valueRed=analogRead(A1)/4;
  if(ktory_kolor==2)
    valueGreen=analogRead(A1)/4;
  if(ktory_kolor==3)
    valueBlue=analogRead(A1)/4;
  analogWrite(LED2RED, valueRed);
  analogWrite(LED2GREEN, valueGreen);
  analogWrite(LED2BLUE, valueBlue);
  wyswietlNaLCD();
  kursor();
  delay(200);

}
```

W programie użyto dwóch funkcji użytkownika `wyswietlNaLCD()` i `kursor()`, w których wyodrębniono instrukcje odpowiedzialne za poprawne wyświetlanie wartości intensywności barw na wyświetlaczu LCD. W funkcji `void loop()` w pierwszej kolejności sprawdzane są stany naciśnięcia przycisków. W zależności od tego, który przycisk jest naciśnięty, zmieniona zostaje wartość zmiennej `ktory_kolor`. Zmienna `ktory_kolor` w kolejnym kroku poddawana jest analizie instrukcjami warunkowymi. Jeśli jej wartość to 1, to z czujnika analogowego A1 pobierana jest wartość wskazania potencjometru i po przeskalowaniu przypisywana jest do zmiennej `valueRed`, na której przechowujemy intensywność koloru czerwonego. Analogiczna instrukcja pojawia się, gdy zmienna `ktory_kolor` ma wartość 2 lub 3 – wówczas ustawiane są zmienne odpowiednio `valueGreen` lub `valueBlue`.

Po ustawieniu wartości zmiennych odpowiedzialnych za intensywność barw wskazane ustawienia intensywności zostają wysłane na piny obsługujące diodę RGB.

W ostatnim kroku następuje wywołanie funkcji `wyswietlNaLCD()` i `kursor()`.

8. Przeprowadzenie quizu podsumowującego lekcję.

1. Który z podanych skrótów nie określa modelu barw
   a) HSV
   b) CSV

c) CMYK
d) RGB

2. Instrukcja wysyłająca dowolną intensywność dla określonej barwy w diodzie RGB to:
   a) `analogRead();`
   b) `lcd.begin();`
   c) `digitalWrite();`
   d) `analogWrite();`
3. Model RGB dla poszczególnych kolorów umożliwia określenie intensywności w zakresie:
   a) Od 0 do 255
   b) Od 1 do 256
   c) Od 0 do 1023
   d) Od 0 do 100
4. Skrót RGB oznacza
   a) Red, Green, Blue
   b) Różony, Granatowy, Niebieski
   c) Różne Warianty Barw
   d) jest skrótem od imion wynalazców

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który wysteruje potencjometrem kolory czerwony, zielony i niebieski na diodzie RGB LED2. Sterowanie powinno polegać na tym, że skrajne lewe wychylenie powinno spowodować włączenie koloru czerwonego na diodzie RGB, pozycja środkowa potencjometru powinna włączyć kolor zielony, a pozycja skrajnie prawa kolor niebieski.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11

void setup() {
 // konfiguracja pinów diody RGB jako pinów wyjściowych
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
}

void loop() {
 if(analogRead(A1)<=(1024*0.33))
 {
    // włącz kolor czerwony
     digitalWrite(LED2RED, HIGH);
     digitalWrite(LED2GREEN, LOW);
     digitalWrite(LED2BLUE, LOW);
 }
  if(analogRead(A1)>(1024*0.33)&&analogRead(A1)<=(1024*0.67))
 {
    // włącz kolor zielony
     digitalWrite(LED2RED, LOW);
     digitalWrite(LED2GREEN, HIGH);
     digitalWrite(LED2BLUE, LOW);
 }
  if(analogRead(A1)>(1024*0.67))
 {
    // włącz kolor niebieski
     digitalWrite(LED2RED, LOW);
     digitalWrite(LED2GREEN, LOW);
```

```
        digitalWrite(LED2BLUE, HIGH);
    }
}
```

Ćwiczenie 2

Napisz program, który wysteruje przyciskami monostabilnymi włączenie bądź wyłączenie kolorów czerwonego, zielonego i niebieskiego na diodzie RGB LED2.

Pojedyncze naciśnięcie przycisku powinno zmieniać stan załączenia koloru, za którego sterowanie dany przycisk jest odpowiedzialny.

— przycisku `SW_LEFT` – koloru czerwonego,
— przycisku `SW_CENTER` – koloru zielonego,
— przycisku `SW_RIGHT` – koloru niebieskiego.

Ćwiczenie 3

Napisz program, który w sposób płynny będzie przechodził między kolorami czerwonym, zielonym i niebieskim. Poszczególne kolory powinny rozświetlać się od 0 do maksymalnej swojej jasności, a następnie powinny się wygasić i przejść w kolejny kolor.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11

void setup() {
 // konfiguracja pinów
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
}

void loop() {
 for(int i=0; i<=255; i++)
 {
   // ustaw jasność diody czerwonej
   analogWrite(LED2RED, i);
   delay(10);
 }
 for(int i=255; i>=0; i--)
 {
   // ustaw jasność diody czerwonej
   analogWrite(LED2RED, i);
   delay(10);
 }
```

```
  for(int i=0; i<=255; i++)
 {
   // ustaw jasność diody zielonej
   analogWrite(LED2GREEN, i);
   delay(10);
 }
 for(int i=255; i>=0; i--)
 {
   // ustaw jasność diody zielonej
   analogWrite(LED2GREEN, i);
   delay(10);
 }

  for(int i=0; i<=255; i++)
 {
   // ustaw jasność diody niebieskiej
   analogWrite(LED2BLUE, i);
   delay(10);
 }
 for(int i=255; i>=0; i--)
 {
   // ustaw jasność diody niebieskiej
   analogWrite(LED2BLUE, i);
   delay(10);
 }
}
```

Ćwiczenie 4

Napisz program, który umożliwi sterowanie kolorem świecenia diody RGB. Sterowanie kolorem świecenie będzie odbywało się za pomocą potencjometru. O tym, który kolor jest w danej chwili sterowany, będzie decydowało naciśnięcie przycisków `SW_LEFT`, `SW_CENTER` i `SW_RIGHT`. `SW_LEFT` włącza sterowanie kolorem czerwonym. `SW_CENTER` włącza sterowanie kolorem zielonym, zaś `SW_RIGHT` włącza sterowanie kolorem niebieskim.

To, który kolor w danym momencie jest regulowany, powinien wskazywać kursor na wyświetlaczu LED, podkreślający odpowiednio litery R (czerwony), G (zielony) lub B (niebieski).

Quiz

1. Który z podanych skrótów nie określa modelu barw
   a) HSV
   b) CSV
   c) CMYK
   d) RGB
2. Instrukcja wysyłająca dowolną intensywność dla określonej barwy w diodzie RGB to:
   a) `analogRead();`
   b) `lcd.begin();`
   c) `digitalWrite();`
   d) `analogWrite();`
3. Model RGB dla poszczególnych kolorów umożliwia określenie intensywności w zakresie:
   a) Od 0 do 255
   b) Od 1 do 256
   c) Od 0 do 1023
   d) Od 0 do 100
4. Skrót RGB oznacza
   a) Red, Green, Blue
   b) Różony, Granatowy, Niebieski
   c) Różne Warianty Barw
   d) jest skrótem od imion wynalazców

Lekcja 10

Losujemy liczby i tworzymy grę „Papier, kamień, nożyce” z użyciem wyświetlacza graficznego OLED w zestawie edukacyjnym TME-EDU-ARD-2.

Czas trwania lekcji: **90 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych oraz pętli w języku C++. Utrwalenie umiejętności odczytu danych z potencjometru oraz przycisków monostabilnych w zestawie edukacyjnym TME-EDU-ARD-2.
Utrwalenie umiejętności sterowania diodą RGB.
Zapoznanie z działaniem funkcji zwracającej liczby pseudolosowe w języku C++.
Zapoznanie z działaniem i możliwościami wyświetlacza graficznego OLED w zestawie edukacyjnym TME-EDU-ARD-2.

Efekty kształcenia

- Uczeń potrafi stosować zmienne w języku C++.
- Uczeń wie jak i kiedy stosuje się pętle i instrukcje warunkowe w języku C++.
- Uczeń stosuje w swoich programach potencjometr oraz przyciski monostabilne do wprowadzania danych.
- Uczeń wie czym są funkcje `random()` i `randomSeed()`.
- Uczeń potrafi przedstawić barwy modelu RGB na diodzie świecącej RGB.
- Uczeń potrafi użyć w programie wyświetlacza OLED.

Wstęp

W trakcie lekcji utrwalimy umiejętności kontroli nad wcześniej poznanymi podzespołami zestawu edukacyjnego.

Ponadto w lekcji zostanie wprowadzona funkcja losująca, dzięki której będziemy mogli symulować wprowadzanie testowych danych (liczb). Tej funkcji użyjemy także do wykonanania prostej gry w „Papier, kamień, nożyce”.

Kolejnym zagadnieniem będzie podstawowa obsługa wyświetlacza graficznego OLED wbudowanego w zestaw edukacyjny.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

1. Jak programować podzespoły poznane na poprzednich lekcjach, tj. przyciski monostabilne, potencjometr i diodę LED?

3. Omówienie przez nauczyciela działania funkcji losującej w układzie Arduino UNO.

Do (pseudo)losowania liczb w programach na układ Arduino UNO służy funkcja `random()`. Funkcja ta posiada możliwość zastosowania dwóch bądź jednego parametru.

Przykłady zastosowań:

```
random(max); // funkcja wylosuje liczbę z zakresu od zera do max-1
random(min,max); // funkcja liczbę z zakresu od min do max-1
```

Aby wylosowane liczby nie były w kolejnych uruchomieniach programu takie same, należy w każdym wykonaniu programu przed pierwszym użyciem funkcji `random()` zainicjalizować generator liczb pseudolosowych za pomocą funkcji `randomSeed(ziarno);`.

Jako parametr (ziarno) funkcji `randomSeed` powinniśmy przyjąć możliwie losową wartość całkowitą. W układzie Arduino UNO najczęściej do tego celu wykorzystuje się odczyt jednego z wejść analogowych. My przyjmiemy jako ziarno wartość odczytaną z wejścia A0, czyli wejścia, do którego w zestawie edukacyjnym jest podłączony mikrofon.

Przykład zainicjowania ziarna:

```
randomSeed(analogRead(0));
```

4. Ćwiczenie 1 Do realizacji wraz z nauczycielem.

Napisz program, który po każdym naciśnięciu przycisku monostabilnego `SW_CENTER` zmieni losowo kolor emitowany przez diodę RGB.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11
#define SW_CENTER 8

void setup() {
 // ustawienie ziarna liczb losowych
 randomSeed(analogRead(0));
 // konfiguracja pinów
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
}
```

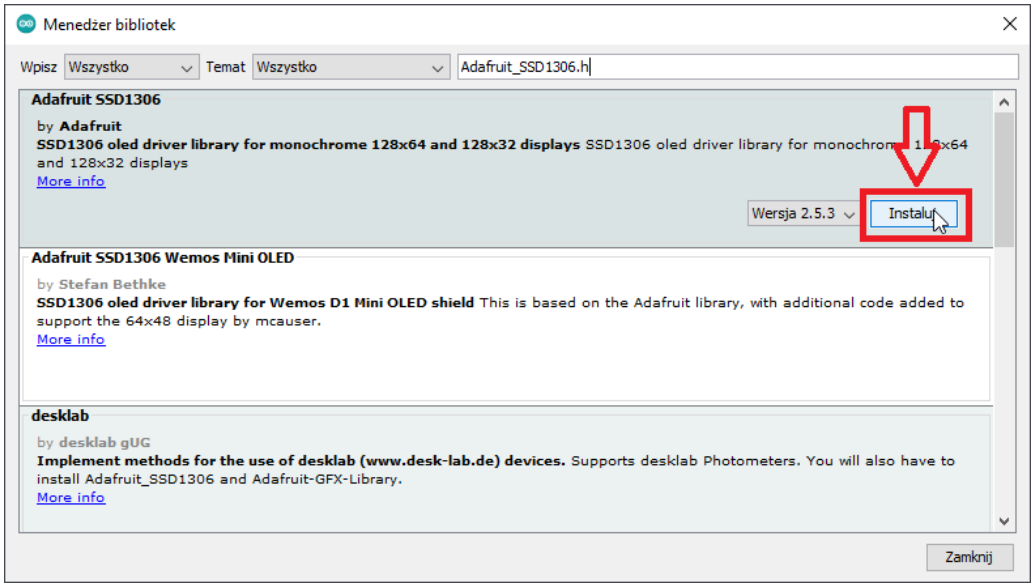
```
void loop() {
  if (digitalRead(SW_CENTER) == HIGH)
  {
    // ustawienie dla każdego koloru diody RGB
    // innej losowej wartości intensywności w zakresie od 0 do 255
    analogWrite(LED2RED, random(256));
    analogWrite(LED2GREEN, random(256));
    analogWrite(LED2BLUE, random(256));
    delay(200);
  }
}
```

W rozwiązaniu w funkcji `void setup()` zainicjowano ziarno generatora liczb pseudolosowych `randomSeed(analogRead(0));`.

Dopiero w funkcji `void loop()` po sprawdzeniu, że przycisk monostabilny `SW_CENTER` jest wciśnięty, ustawiany jest poziom intensywności kolorów diody RGB. W funkcji `analogWrite()` zastosowano funkcję `random()` z jednym parametrem. Parameter ten po pomniejszeniu o 1 określa zakres maksymalny wylosowanych wartości licząc od 0.

5. Omówienie przez nauczyciela podstawowych instrukcji i bibliotek stosowanych przy sterowaniu wyświetlaczem graficznym OLED.

By skorzystać z funkcji wyświetlacza graficznego OLED, koniecznym jest zastosowanie bibliotek Adafruit_SSD1306.h, Adafruit_GFX.h i Wire.h. Musimy sprawdzić, czy te biblioteki są zainstalowane w naszym ArduinoIDE. Wchodzimy więc do Menedżera bibliotek i wyszukujemy biblioteki po nazwie. Jeśli np. pozycja „Adafruit SSD1206” nie jest zainstalowana, klikamy przycisk „Instaluj”.



Zestawienie podstawowych poleceń wyświetlacza graficznego:

— `Adafruit_SSD1306 display(NULL);` // tworzymy obiekt wyświetlacza, wykonujemy to przed wszystkimi funkcjami

- `display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);` // przed pierwszym użyciem konfiguruje podstawowe dane o wyświetlaczu
- `display.clearDisplay();`
- `// czyści wyświetlacz`
- `display.setTextColor(WHITE);` // ustawia kolor czcionki na biały(jedyny możliwy kolor)
- `display.setCursor(poziom,pion);` // ustawia pozycję kursora
- `display.println("Hello");` // wyświetla na ekranie przekazany łańcuch danych
- `display.drawBitmap(0, 0, tablica_bitmapy, 128, 64, 1);` // wysyła na wyświetlacz tablicę z bitmapą obrazu. Parametr `0, 0,` oznacza pozycję lewego górnego rogu ekranu. Parametr `128, 64` oznacza wielkość bitmapy. Ostatni parametr to kolor (1=biały).
- `display.display();` // wysyła wprowadzone ustawienia na wyświetlacz (aktualizuje dane na wyświetlaczu)

6. Ćwiczenie 2 Do realizacji wraz z nauczycielem.

Napisz program, który na wyświetlaczu graficznym OLED wypisze tekst „Hello World!!!”.

Tekst powinien być napisany białą czcionką.

Spróbuj tak ustalić pozycję kursora, by każde słowo wypisane było w osobnym wierszu i możliwie jak najbliżej środka ekranu.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(NULL);

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.clearDisplay();
 display.setTextColor(WHITE);
 display.setCursor(50,3);
 display.println("Hello");
 display.setCursor(40,16);
 display.println("World!!!");
 display.display();
}

void loop() {
}
```

Na początku wczytujemy bibliotekę `Adafruit_SSD1306.h`. Biblioteka ta wystarczy do podstawowych działań na tekstach wyświetlanych na wyświetlaczu graficznym. Do dalszej pracy tworzymy obiekt wyświetlacza funkcją `Adafruit_SSD1306 display(NULL);`.

W funkcji `void setup()` ustawiamy ziarno losowania `randomSeed(analogRead(0));` i teraz możemy dokonać ustawienia wyświetlacza funkcjami `display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);` i `display.setTextColor(WHITE);`.

Po wyczyszczeniu ekranu funkcją `display.clearDisplay();` ustawiamy kursor w dobranej optymalnej dla słowa „Hello” pozycji, tak by był on wyświetlony środkowo. `display.setCursor(50,3);`. Wyświetlenia słowa „Hello” dokonamy za pomocą instrukcji `display.println("Hello");`.

Podobnie jak dla słoa „Hello” postępujemy przy wysłaniu do wyświetlacza słowa „World!!!”.

By zobaczyć efekt naszej konfiguracji tekstu na wyświetlaczu wykonujemy polecenie `display.display();`.

7. Ćwiczenie 3 Do realizacji samodzielnie przez uczniów.

Zmodyfikuj program z Ćwiczenia 2 tak, by napis wyświetlany był na losowej wysokości wyświetlacza. Położenie pionowe powinno się zmieniać przy każdym naciśnięciu przycisku `SW_CENTER`.

Ponadto pozioma pozycja tekstu powinna być regulowana potencjometrem, tzn., jeżeli potencjometr jest w pozycji skrajnie lewej, to tekst powinien znajdować się w skrajnie lewej pozycji wyświetlacza, a jeśli potencjometr jest skrajnie po prawej, to tekst powinien znajdować się w prawej skrajnej pozycji wyświetlacza.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>

#define SW_CENTER 8

Adafruit_SSD1306 display(NULL);

void setup() {
 // ustawienie ”ziarna” liczb losowych
 randomSeed(analogRead(0));
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.setTextColor(WHITE);
}

int wysokosc=0;
int szerokosc=0;
void loop() {
   display.clearDisplay();
   if (digitalRead(SW_CENTER) == HIGH)
   {
    // przypisanie losowej wartości położenia pionowego dla napisu
     wysokosc=random(18);
   }
   // przypisanie poziomej wartości położenia napisu
   szerokosc=analogRead(A1)/(1023/77);
   display.setTextColor(WHITE);
   display.setCursor(szerokosc+10,wysokosc);
   display.println("Hello");
   display.setCursor(szerokosc,wysokosc+8);
```

```
        display.println("World!!!");
        display.display();
    }
```

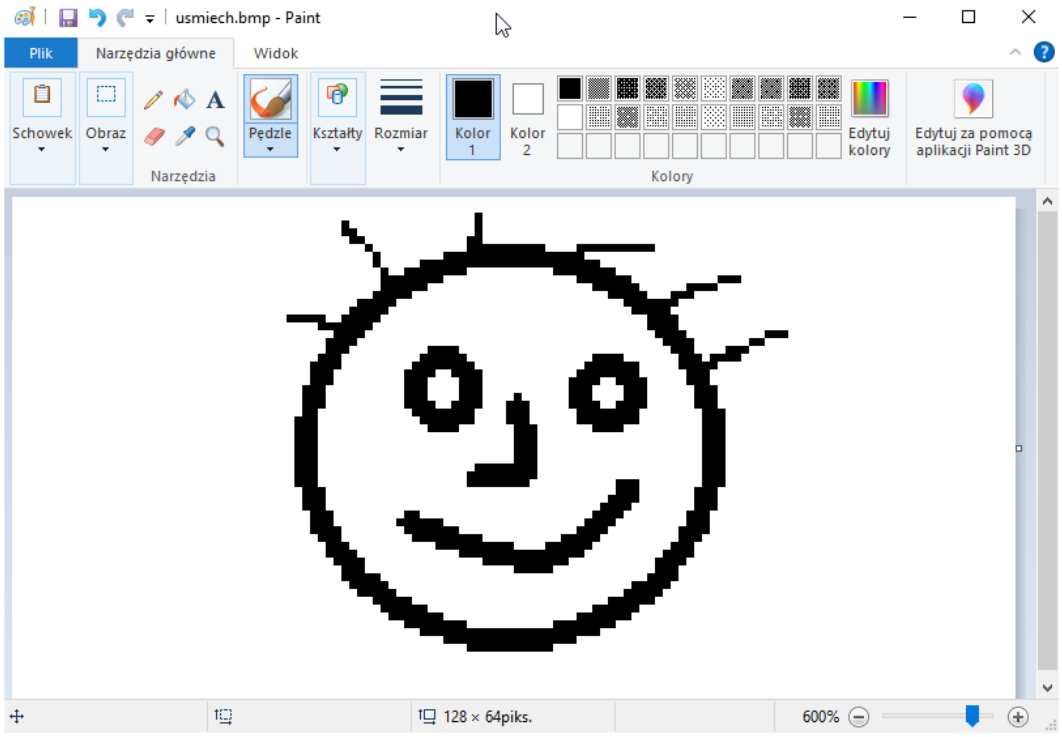
Instrukcje konfiguracyjne wyświetlacza w tym ćwiczeniu są podobne do tych z ćwiczenia 2.

W funkcji `void loop()`, by ustalić poziom (położenie w pionie) wyświetlanych napisów, losujemy liczbę z zakresu od 0 do 17 (wartości skrajne dobrano tak, by zawsze był widoczny cały napis „Hello World!!!”).

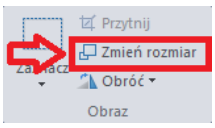
Przy ustalaniu pozycji w poziomie pomocny jest nam odczyt z wejścia analogowego A1, do którego podpięty jest w zestawie edukacyjnym potencjometr.

8. Omówienie przez nauczyciela sposobu przygotowania pliku graficznego do wyświetlenia na wyświetlaczu graficznym OLED

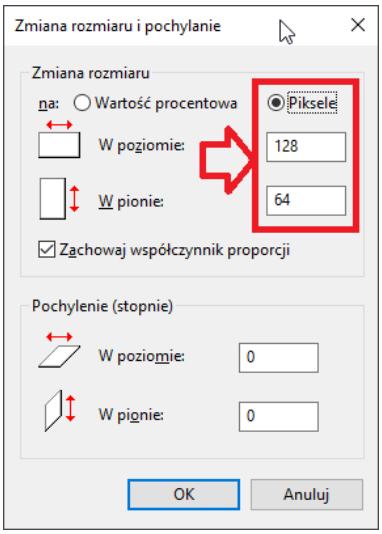
Aby przygotować plik do wyświetlenia na wbudowanym wyświetlaczy OLED, musimy pozsłużyć się kilkoma dodatkowymi programami. Pierwszy z nich to program graficzny (np. MS Paint), w którym przygotujemy grafikę w formacie BMP, w rozdzielczości 128×64 piksele. Taką właśnie rozdzielczość ma nasz wyświetlacz OLED.



Wprogramie MS Paint, by ustaić rozmiar pliku na 128×64 px, wystarczy w narzędziach głównych wybrać opcję „Zmień rozmiar”.

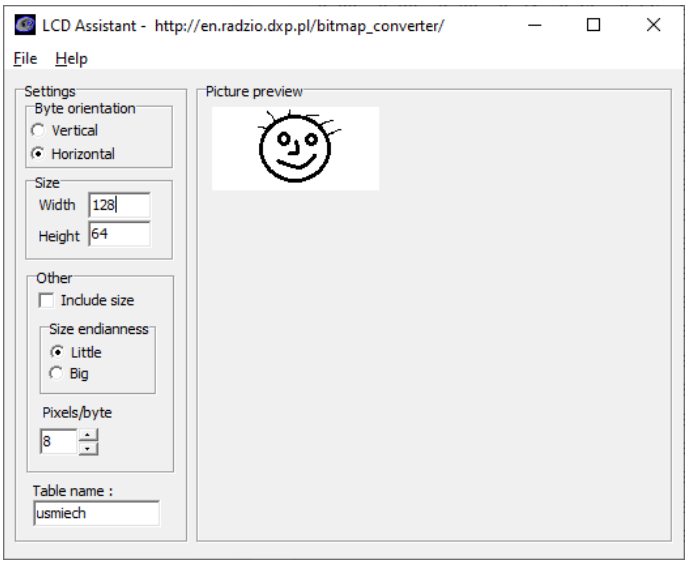


Następnie ustalić jednostkę: „piksele” i ustawić wymiary na 128 i 64 px.



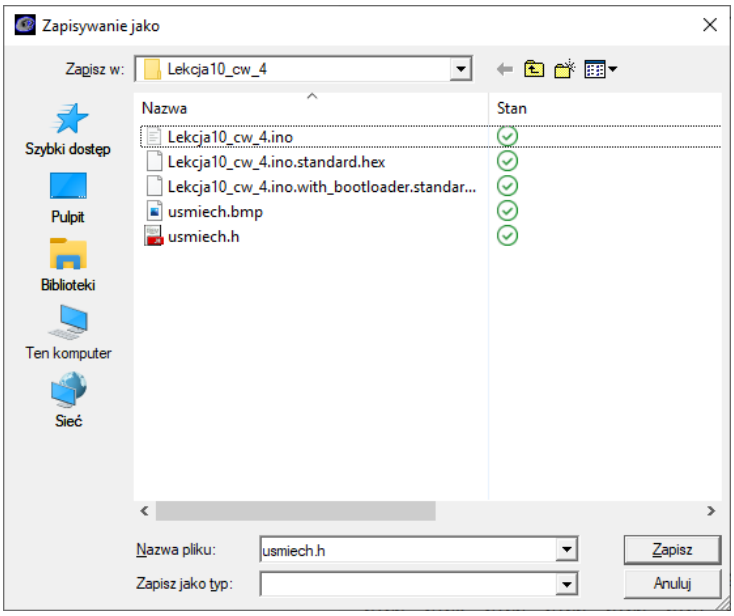
Kolejnym krokiem jest konwersja bitmapy w formacie BMP na tablicę danych dla programu w języku C++.

Do tego celu proponujemy wykorzystać program LCD Assistant. Jest to darmowy program do ściągnięcia ze strony `http://en.radzio.dxp.pl/bitmap_convert/`. Program jest typu portable, czyli nie wymaga instalacji. Uruchamiamy więc plik exe i wczytujemy przygotowaną wcześniej bitmapę.



Przed konwersją ustawiamy opcje konwersji zgodnie z przykładem załączonym powyżej.

Następnie zapisujemy skonwertowany plik, dodając mu rozszerzenie „.h”. Takie rozszerzenie ułatwi nam dołączenie pliku i tabeli z obrazem do naszego programu w C++.



Na koniec otwórzmy plik w dowolnym edytorze tekstowym np. Notatnik lub Notepad++, i w nagłówku tabeli dodajmy właściwość `PROGMEM`.

Przykładowy nagłówek obrazu BMP o nazwie „uśmiech” przekonwertowanego do tablicy danych:

```
const unsigned char usmiech [] PROGMEM= {
```

9. Ćwiczenie 4 Do realizacji wraz z nauczycielem.

Napisz program, który na wyświetlaczu graficznym OLED wyświetli znak monochromatyczny (czarno-biały) zapisany w pliu BMP.

Przykładowy kod programu programu głównego (roszerzenie „.ino”):

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>

// wczytanie pliku nagłówkowego zawierającego tabicę kodową plku BMP
#include "usmiech.h"

Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);

void setup(){
  display.begin(0x2, 0x3C, false);
  display.clearDisplay();
  // wczytanie bitmapy z tablicy usmiech (z pliku usmiech.h)
```

```
  display.drawBitmap(0, 0, usmiech, 128, 64, 1);
  display.display();
}

void loop(){
}
```

Przykładowy fragment kodu pliku nagłówkowego z tablicą przekonwertowanej bitmapy o nazwie usmiech (roszerzenie „.ino”):

```
const unsigned char usmiech [] PROGMEM= {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
.....
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00
};
```

W pierwszej kolejności wczytujemy biblioteki konieczne do obsługi wyświetlacza OLED.

I definiujemy stałe do konfiguracji parametrów wyświetlacza.

Po przygotowaniu bitmapy i przekonwertowaniu do tablicy danych musimy dołączyć plik nagłówkowy, w którym ta tablica się znajduje.

Kolejny krok to utworzenie obiektu wyświetlacza za pomocą instrukcji `Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire, RESET_BTN);`. W funkcji `void setup()` dokonujemy konfigurację wyświetlacza `display.begin(0x2, 0x3C, false);`. Czyścimy ekran funkcją `display.clearDisplay();`. Możemy teraz wysłać nasz obraz(tablicę z przekonwertowaną bitmapą o nazwie uśmiech) `display.drawBitmap(0, 0, usmiech, 128, 64, 1);`. Ostatnie polecenie, które powoduje wyświetlenie ustalonej konfiguracji to `display.display();`.

10. Ćwiczenie 5 Do realizacji przez uczniów.

Napisz program, który będzie symulował grę w „Papier, kamień, nożyce”.

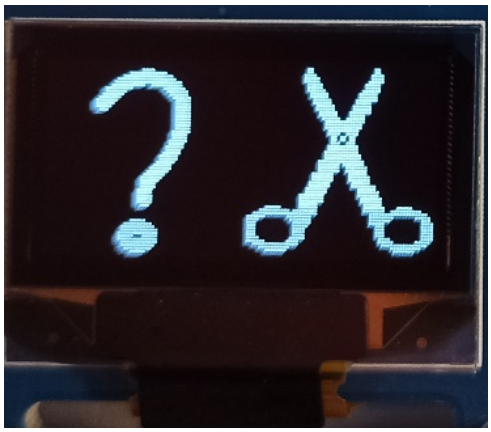
Program powinien umożliwiać wybranie przez użytkownika opcji(stanowiska) do gry. Wybór użytkownika powinien odbywać się za pomocą potencjometru.

Skrajne lewa pozycja potencjometru to „Papier”.
Środkowa pozycja potencjometru to „Kamień”.
Skrajnie prawa pozycja potencjometru to „Nożyce”.

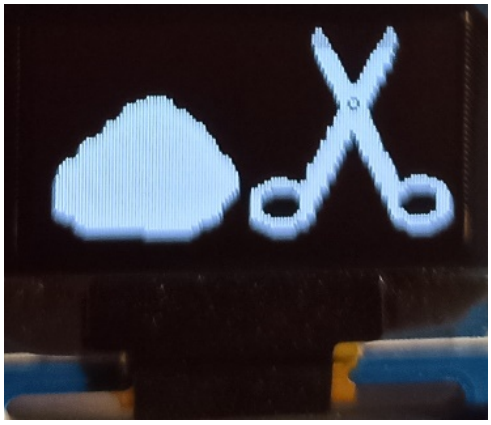
Wybrana opcja powinna być wyświetlona na ekranie OLED.
Zakończenie (potwierdzenie) wyboru opcji (stanowiska) użytkownika powinno odbywać się przez przycisk `SW_CENTER`.

Po zatwierdzeniu na wyświetlaczu powinien pokazywać się również wybór mikrokontrolera układu Arduino UNO.

Przykładowy ekran prezentujący wybieranie opcji użytkownika (symbol z prawej strony):



Przykładowy ekran prezentujący wybór mikrokontrolera ArduinoUNO („Kamień”) i użytkownika „Nożyce”:



Przykładowy kod źródłowy rozwiązania ćwiczenia:

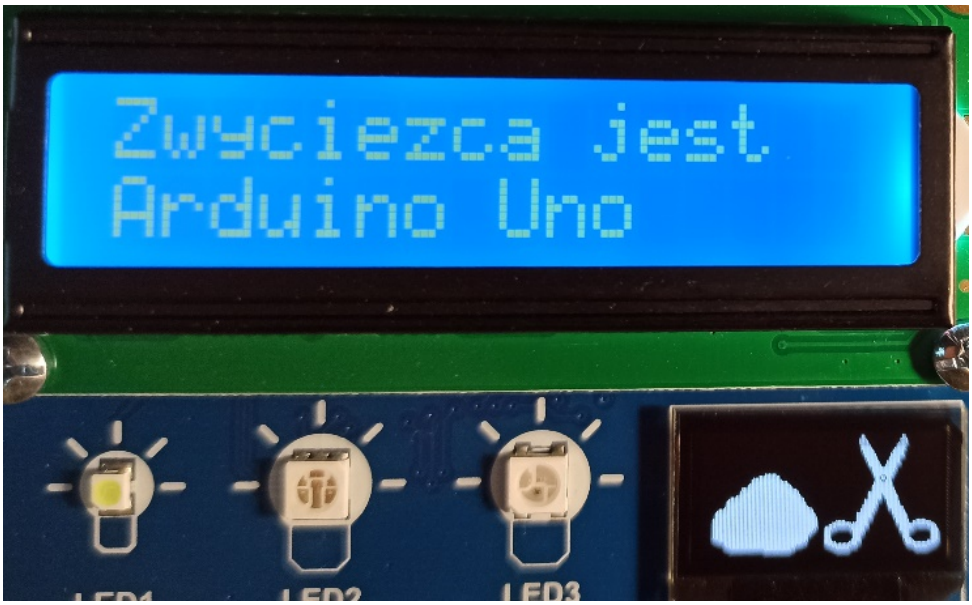
```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>

// wczytanie pliku nagłówkowego zawierającego tabicę kodową plku BMP
#include "papier.h"
#include "kamien.h"
#include "nozyce.h"
#include "niewiadoma.h"

#define SW_CENTER 8

int wybor;
int los;
Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);
```

```
void setup(){
  // ustawienie ziarna liczb losowych
  randomSeed(analogRead(0));

  display.begin(0x2, 0x3C, false);
  los=random(1,3);
  while((digitalRead(SW_CENTER) == LOW))
  {
    wybor=((analogRead(A1)-1)/(1023/3))+1;
    display.clearDisplay();
    display.drawBitmap(0, 0, niewiadoma, 64, 64, 1);
    if(wybor==1)
      display.drawBitmap(64, 0, papier, 64, 64, 1);
    if(wybor==2)
      display.drawBitmap(64, 0, kamien, 64, 64, 1);
    if(wybor==3)
      display.drawBitmap(64, 0, nozyce, 64, 64, 1);
    display.display();
  }
  display.clearDisplay();
    if(los==1)
      display.drawBitmap(0, 0, papier, 64, 64, 1);
    if(los==2)
      display.drawBitmap(0, 0, kamien, 64, 64, 1);
    if(los==3)
      display.drawBitmap(0, 0, nozyce, 64, 64, 1);
    if(wybor==1)
      display.drawBitmap(64, 0, papier, 64, 64, 1);
    if(wybor==2)
      display.drawBitmap(64, 0, kamien, 64, 64, 1);
    if(wybor==3)
      display.drawBitmap(64, 0, nozyce, 64, 64, 1);
    display.display();
}

void loop(){
}
```

11. Ćwiczenie 6 Do realizacji przez uczniów.

Do realizacji z Ćwiczenia 5 dodaj sprawdzenie zwycięzcy. Informację o tym, kto jest zwycięzcą rozgrywki podaj na wyświetlaczu LCD.

Przykładowy ekran LCD i OLED podsumowujący rozgrywkę:



Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// wczytanie pliku nagłówkowego zawierającego tabicę kodową plku BMP
#include "papier.h"
#include "kamien.h"
#include "nozyce.h"
#include "niewiadoma.h"

#define SW_CENTER 8

int wybor;
int los;
Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);

void setup(){
  // ustawienie ”ziarna” liczb losowych
  randomSeed(analogRead(0));
```

```
  // konfiguracja LCD
  lcd.begin(16, 2);
  display.begin(0x2, 0x3C, false);
  los=random(1,3);
  while((digitalRead(SW_CENTER) == LOW))
  {
    wybor=((analogRead(A1)-1)/(1023/3))+1;
    display.clearDisplay();
    display.drawBitmap(0, 0, niewiadoma, 64, 64, 1);
    if(wybor==1)
      display.drawBitmap(64, 0, papier, 64, 64, 1);
    if(wybor==2)
      display.drawBitmap(64, 0, kamien, 64, 64, 1);
    if(wybor==3)
      display.drawBitmap(64, 0, nozyce, 64, 64, 1);
    display.display();
  }
  display.clearDisplay();
    if(los==1)
      display.drawBitmap(0, 0, papier, 64, 64, 1);
    if(los==2)
      display.drawBitmap(0, 0, kamien, 64, 64, 1);
    if(los==3)
      display.drawBitmap(0, 0, nozyce, 64, 64, 1);
    if(wybor==1)
      display.drawBitmap(64, 0, papier, 64, 64, 1);
    if(wybor==2)
      display.drawBitmap(64, 0, kamien, 64, 64, 1);
    if(wybor==3)
      display.drawBitmap(64, 0, nozyce, 64, 64, 1);
    display.display();

   lcd.clear();
  if((wybor==2 && los==1)||(wybor==3 && los==2)||(wybor==1 &&
los==3))
  {
    lcd.setCursor(0, 0);
    lcd.print("Zwyciezca jest");
    lcd.setCursor(0, 1);
    lcd.print("Arduino Uno");

  }
  if((wybor==1 && los==2)||(wybor==2 && los==3)||(wybor==3 &&
los==1))
  {
    lcd.setCursor(0, 0);
```

```
        lcd.print("JESTES");
        lcd.setCursor(0, 1);
        lcd.print("WYGRANYM!!!");
      }
      if(wybor==los)
      {
        lcd.setCursor(0, 0);
        lcd.print("JEST");
        lcd.setCursor(0, 1);
        lcd.print("REMIS");
      }

}

void loop(){
}
```

12. Przeprowadzenie quizu podsumowującego lekcję.

1. Rozdzielczość wyświetlacza wbudowanego w zestaw edukacyjny TME-EDU-ARD-2 to:
   a) 128 pikseli szerokości i 128 pikseli wysokości
   b) 128 pikseli szerokości i 64 pikseli wysokości
   c) 64 pikseli szerokości i 128 pikseli wysokości
   d) 64 pikseli szerokości i 64 pikseli wysokości
2. Do wylosowania wartości w zakresie od 0 do 100 włącznie użyjemy funkcji:
   a) randomSeed(0,100);
   b) randomSeed(0,101);
   c) random(100);
   d) random(101);
3. Obraz z przekonwertowanego pliku BMP wyświetlimy na wyświetlaczu za pomocą polecenia:
   a) display.drawBitmap()
   b) display()
   c) display.clearDisplay()
   d) display.print();
4. Wskaż niepoprawnie zapisaną instrukcję
   a) display.clearDisplay();
   b) display.begin(0x2, 0x3C, false);
   c) display.setCursor(50,3);
   d) display.drawBitmap(0, 0, “usmiech.bmp”, 128, 64, 1);

Prawidłowe odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który po każdym naciśnięciu przycisku monostabilnego `SW_CENTER` zmieni losowo kolor emitowany przez diodę RGB.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11
#define SW_CENTER 8

void setup() {
 // ustawienie ziarna liczb losowych
 randomSeed(analogRead(0));
 // konfiguracja pinów
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
}

void loop() {
  if (digitalRead(SW_CENTER) == HIGH)
  {
    // ustawienie dla każdego koloru diody RGB
    // innej losowej wartości intensywności w zakresie od 0 do 255
    analogWrite(LED2RED, random(256));
    analogWrite(LED2GREEN, random(256));
    analogWrite(LED2BLUE, random(256));
    delay(200);
  }
}
```

Ćwiczenie 2

Napisz program, który na wyświetlaczu graficznym OLED wypisze tekst „Hello World!!!”.

Tekst powinien być napisany białą czcionką.

Spróbuj tak ustalić pozycję kursora, by każde słowo wypisane było w osobnym wierszu i możliwie jak najbliżej środka ekranu.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(NULL);

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.clearDisplay();
 display.setTextColor(WHITE);
 display.setCursor(50,3);
 display.println("Hello");
 display.setCursor(40,16);
 display.println("World!!!");
 display.display();
}

void loop() {
}
```

Ćwiczenie 3

Zmodyfikuj program z Ćwiczenia 2 tak, by napis wyświetlany był na losowej wysokości wyświetlacza. Położenie pionowe powinno się zmieniać przy każdym naciśnięciu przycisku `SW_CENTER`.

Ponadto pozioma pozycja tekstu powinna być regulowana potencjometrem, tzn., jeżeli potencjometr jest w pozycji skrajnie lewej, to tekst powinien znajdować się w skrajnie lewej pozycji wyświetlacza, a jeśli potencjometr jest skrajnie po prawej, to tekst powinien znajdować się w prawej skrajnej pozycji wyświetlacza.

Ćwiczenie 4

Napisz program, który na wyświetlaczu graficznym OLED wyświetli znak monochromatyczny (czarno-biały) zapisany w pliu BMP.

Przykładowy kod programu programu głównego (roszerzenie „.ino”):

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>

// wczytanie pliku nagłówkowego zawierającego tabicę kodową plku BMP
#include "usmiech.h"
```

```
Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);

void setup(){
  display.begin(0x2, 0x3C, false);
  display.clearDisplay();
  // wczytanie bitmapy z tablicy usmiech (z pliku usmiech.h)
  display.drawBitmap(0, 0, usmiech, 128, 64, 1);
  display.display();
}

void loop(){
}
```

Ćwiczenie 5

Napisz program, który będzie symulował grę w „Papier, kamień, nożyce”.

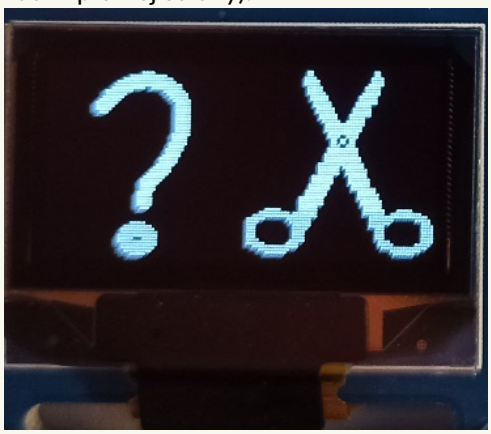
Program powinien umożliwiać wybranie przez użytkownika opcji(stanowiska) do gry. Wybór użytkownika powinien odbywać się za pomocą potencjometru.

Skrajne lewa pozycja potencjometru to „Papier”.
Środkowa pozycja potencjometru to „Kamień”.
Skrajnie prawa pozycja potencjometru to „Nożyce”.

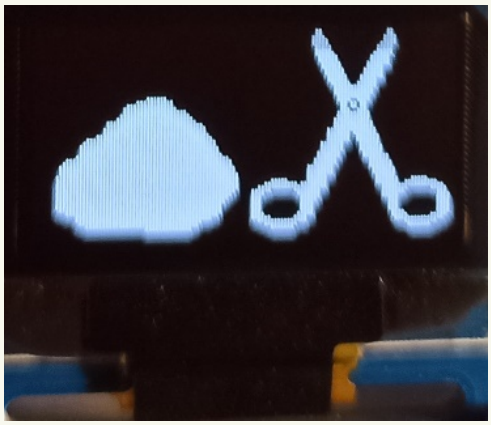
Wybrana opcja powinna być wyświetlona na ekranie OLED.
Zakończenie (potwierdzenie) wyboru opcji (stanowiska) użytkownika powinno odbywać się przez przycisk `SW_CENTER`.

Po zatwierdzeniu na wyświetlaczu powinien pokazywać się również wybór mikrokontrolera układu Arduino UNO.

Przykładowy ekran prezentujący wybieranie opcji użytkownika (symbol z prawej strony):



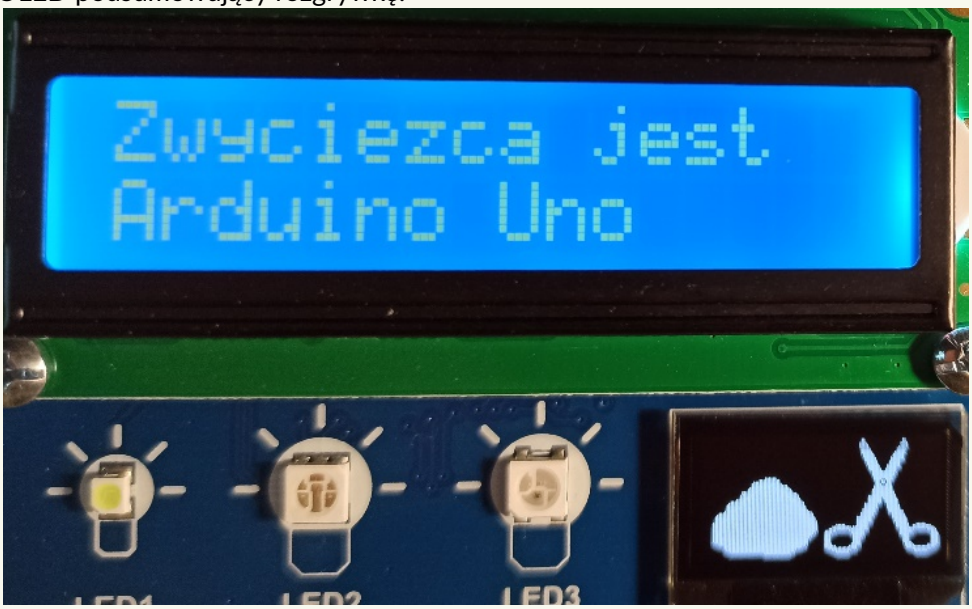
Przykładowy ekran prezentujący wybór mikrokontrolera ArduinoUNO („Kamień”) i użytkownika „Nożyce”:



Ćwiczenie 6

Do realizacji z Ćwiczenia 5 dodaj sprawdzenie zwycięzcy. Informację o tym, kto jest zwycięzcą rozgrywki podaj na wyświetlaczu LCD.

Przykładowy ekran LCD i OLED podsumowujący rozgrywkę:



Quiz

1. Rozdzielczość wyświetlacza wbudowanego w zestaw edukacyjny TME-EDU-ARD-2 to:
   a) 128 pikseli szerokości i 128 pikseli wysokości
   b) 128 pikseli szerokości i 64 pikseli wysokości
   c) 64 pikseli szerokości i 128 pikseli wysokości
   d) 64 pikseli szerokości i 64 pikseli wysokości
2. Do wylosowania wartości w zakresie od 0 do 100 włącznie użyjemy funkcji:
   a) randomSeed(0,100);
   b) randomSeed(0,101);
   c) random(100);
   d) random(101);
3. Obraz z przekonwertowanego pliku BMP wyświetlimy na wyświetlaczu za pomocą polecenia:
   a) display.drawBitmap()
   b) display()
   c) display.clearDisplay()
   d) display.print();
4. Wskaż niepoprawnie zapisaną instrukcję
   a) display.clearDisplay();
   b) display.begin(0x2, 0x3C, false);
   c) display.setCursor(50,3);
   d) display.drawBitmap(0, 0, “usmiech.bmp”, 128, 64, 1);

Lekcja 11

Szyfrujemy wiadomości tekstowe szyfrem podstawieniowym Cezara. Wykorzystujemy funkcję `switch()`.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych i pętli w języku C++.
Utrwalenie umiejętności odczytu danych z potencjometru oraz przycisków monostabilnych w zestawie edukacyjnym TME-EDU-ARD-2.
Utrwalenie umiejętności pisanai programów działających z wyświetlaczami LCD, graficznym OLED i LCD.
Utrwalenie umiejętności stosowanai w programach komunikacji UART.
Zapoznanie z algorytmem szyfrowania szyfrem podstawieniowym Cezara.

Efekty kształcenia

1. Uczeń potrafi stosować zmienne w języku C++.
2. Uczeń wie jak działa komunikacja UART.
3. Uczeń stosuje w swoich programach potencjometr oraz przyciski monostabilne do wprowadzania danych.
4. Uczeń stosuje w swoich programach wyświetlacze LCD, graficzny OLED i LED.
5. Uczeń szyfruje i odszyfrowuje kominikaty za pomocą szyfru Cezara.
6. Uczeń stosuje w swoich programach funkcję `switch()`.

Wstęp

Lekcja będzie okazją do utrwalenia umiejętności programowania wyświetlaczy LCD, LED, OLED dostępnych w zestawie edukacyjnym.

Poznamy czym jest szyfr Cezara i jak możemy zaimplementować ten szyfr w programach z Arduino UNO.

Lekcja będzie również okazją do utrwalenia umiejętności dzielenia programów na funkcje własne użytkownika, które będą wywoływane w zależności od wybranej opcji MENU za pomocą funkcji języka C++ `switch()`.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

1. Jak programować poznane na poprzednich lekcjach podzespoły umożliwiające wprowadzanie danych, tj. przyciski monostabilne, potencjometr.
2. Powtórzenie wiadomości o sposobie sterowania i wyświetlania danych na wyświetlaczu LCD, LED i wyświetlaczu graficznym OLED.

3. Omówienie przez nauczyciela działania szyfru podstawieniowego Cezara.

Szyfr Cezara to szyfr, w którym pod każdą literę tekstu jawnego (wejściowego) podstwiamy literę, która w alfabecie angielskim jest przesunięta o ustaloną liczbę liter do przodu względem litery z tekstu jawnego. Liczba liter, o jaką należy przesunąćliterę tekstu jawnego w alfabecie to tzw. **klucz**.

klucz

Wartość klucza powinna być znana nadawcy i odbiorcy wiadomości i powinna być taka sama dla obu z nich.

Jeśli zamieniana litera z tekstu jawnego po przesunięciu o wartość klucza będzie wychodziła dalej niż litera „Z”, należy kontynuować przesunięcie od początku alfabetu.

Przykład dla wartości klucza 5.

Litera tekstu jawnego „D” to po zamianie litera „I”.

Litera tekstu jawnego „W” to po zamianie litera „B”.

Innymi słowy, literom łacińskim (alfabetu 26-literowego) przypisujemy wartości liczbowe od `"A"` = 1 do `"Z"` = 26. Zaszyfrowanie kluczem (liczbą) `k` to dodanie do tak interpretowanej litery-liczby wartości klucza (liczby `k`) modulo 26, czyli w przypadku, gdy uzyskana wartość przekracza 26 – pomniejszeniu jej o tę liczbę.

Odszyfrowanie przebiega podobnie, tylko litera z szyfrogramu jest cofana o wartość klucza. Jeśli litera z szyfrogramu po pomniejszeniu o klucz jet mniejsza niż „A” (otrzymaliśmy liczbę ujemną lub 0), to cofanie kontynuujemy od końca alfabetu.

UWAGA! Podstawienie Cezara jest szyfrem bardzo łatwym do złamania, nie należy posługiwać się nim tam, gdzie zależy nam na zachowaniu poufności informacji!
Używa się go do celów edukacyjnych, jak tutaj, lub do przekazania odpowiedzi do zadania itp. tak, by nie „spoilować” tym, którzy chcą sami rozwiązać, a zarazem podać odpowiedź.
Podstawienie Cezara jest błyskotliwym przykładem „security by obscurity”, o doniosłych konsekwencjach historycznych: rozkazy i in. poufne wiadomości Juliusza Cezara były bezpieczne, ponieważ oprócz wtajemniczonych nikt nie wpadł na pomysł, jak prosta jest w istocie metoda szyfrowania. Wiedząc, że wiadomość jest zapisana podstawieniem Cezara, przy odrobinie wprawy można ją rozkodować nawet w pamięci.

Podstawienie Cezara nie jest szyfrowaniem w dzisiejszym rozumieniu tego słowa.

Jeżeli wiadomość jest odpowiednio długa, to można ją odkodować patrząc na częstość poszczególnych liter.

Jeżeli wiemy, że wiadomość „zaszyfrowano” podstawieniem Cezara, to wystarczy sprawdzić co najwyżej 26 kluczy i mamy ten właściwy – w tym sensie nie musimy go znać przed „pracą z szyfrogramem”. Wiedząc, że tekst został zakodowany podstawieniem Cezara, a nie znając klucza, można po prostu wielokrotnie poddawać go podstawieniu Cezara z kluczem 1: n złożeń podstawienia Cezara o kluczu 1 to podstawienie Cezara o kluczu n.

Dla zaawansowanych: Łatwo widzieć, że pewna wartość klucza pozwala użyć dokładnie tego samego programu do zaszyfrowania jak i odszyfrowania wiadomości. (Jaka? – odpowiedź: 13.) Dla tej wartości klucza, „szyfrując” tekst już zaszyfrowany, odszyfrujemy go. (Por. symetria osiowa w geometrii płaskiej – przykład przekształcenia, które złożone z samym sobą daje identyczność.) Własność, że to samo działanie szyfruje tekst jawny, a odszyfrowuje zaszyfrowany, posiada też szyfrowanie poprzez XOR bitów tekstu z bitami klucza, metoda uważana za bezpieczną, o ile klucz spełnia określone warunki.

/+ UWAGA! Podstawienie Cezara jest szyfrem bardzo łatwym do złamania, nie należy używać go tam, gdzie zależy nam na poufności informacji! To, do czego nadaje się w praktyce, to ukrycie odpowiedzi do zagadki tak, by zainteresowani mogli ją odczytać, a zainteresowani rozwiązaniem zagadki nie mieli „spoilera”

4. Ćwiczenie 1 Do realizacji wraz z nauczycielem.

Napisz program, który zaszyfruje szyfrem Cezara tekst przesłany za pomocą transmisji UART. Zaszyfrowany tekst zostanie wyświetlony na ekranie OLED.

Do realizacji ćwiczenia zdefinuj w programie stałą KLUCZ. Testowo nadaj jej wartość 4.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>

#define KLUCZ 4

string cezar;

Adafruit_SSD1306 display(NULL);

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.setTextColor(WHITE);
 // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
}

void loop() {
  while(Serial.available() == 0){}
  if(Serial.available() > 0)
  {
    cezar = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
  }
  for(int i=0; i<cezar.length(); i++)
```

```
  {
// sprawdzamy, czy podany znak to litera:
if(cezar[i]>='A' && cezar[i]<='Z')

{
        // jeżeli litera po zwiększeniu o wartość klucza jest większa niż 'Z'
        // to przesuń się do początku alfabetu, czyli o 26 znaków wstecz
       if(cezar[i]+KLUCZ>'Z')
        cezar[i]-=26;

       cezar[i]+=KLUCZ;
    }
  }
   display.clearDisplay();
   display.setTextColor(WHITE);
   display.setCursor(0,0);
   display.println(cezar);
   display.display();
}
```

W programie po wstępnej konfiguracji układu w funkcji `void setup()` przechodzimy do sprawdzania stanu buforu portu szeregowego `while(Serial.available() == 0){}`.

Jeśli w porcie szeregowym odbierzemy jakieś znaki, są one wczytywane do zmiennej `string cezar` za pomocą instrukcji `cezar = Serial.readStringUntil('\n');`.

W kolejnym kroku analizujemy litera po literze znaki wprowadzone z komputera. Sprawdzamy, czy wprowadzony znak należy do przedziału od 'A' do 'Z'. Jeśli tak, to sprawdzamy, czy po powiększeniu o wartość stałej `KLUCZ` nie przekracza 26, czyli wartości litery 'Z'. Jeśli tak, to cofamy znak o 26, czyli o liczbę liter w alfabecie angielskim (łacińskim).

Na koniec wysyłamy zakodowany `string cezar` na wyświetlacz OLED.

5. Ćwiczenie 2 Do realizacji samodzielnej przez uczniów.

Napisz program, który zakoduje podstawieniem Cezara tekst przesłany za pomocą transmisji UART. Zakodowany tekst zostanie wyświetlony na ekranie OLED.

Do realizacji ćwiczenia wczytaj klucz, który pobierzsz jako wskazanie wartości potencjometru. Przyjmij zakres wartości klucza od 0 do 9.

Wyświetl wartość klucza na wyświetlaczu LED.

Dla dociekliwych: Po co zezwalamy na wartość klucza 0, jeżeli pozostawia on wiadomość niezaszyfrowaną (niezmienioną)? Wartość klucza 0 służy np. do sprawdzenia poprawności działania programu lub wyświetlacza.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED
#include "Adafruit_MCP23008.h"// biblioteka LED

int klucz;
```

```
string cezar;
// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

// utworzenie obiektu wyświetlacza LED
Adafruit_MCP23008 seg7;

// utworzenie tablicy z cyframi dla wyświetlacza LED
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.setTextColor(WHITE);
 // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
   // konfiguracja pinów ekspandera wyświetlacza LED
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
}

void loop() {
  while(Serial.available() == 0)
  {
    klucz=analogRead(A1)/(1023/10);
    seg7.writeGPIO(digit[klucz]);
  }

  if(Serial.available() > 0)
  {
```

```
    cezar = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
  }

  for(int i=0; i<cezar.length(); i++)
  {
    // poniżej znajduje się sprawdzenie czy znalizowany znak to litera
    if(cezar[i]>='A' && cezar[i]<='Z')
    {
    // jeżeli litera po zwiększeniu o wartość klucza jest większa niż 'Z'
    // to przesuń się do początku alfabetu, czyli o 26 znaków wstecz
      if(cezar[i]+klucz>'Z')
       cezar[i]-=26;

      cezar[i]+=klucz;
    }
  }
   display.clearDisplay();
   display.setTextColor(WHITE);
   display.setCursor(0,0);
   display.println(cezar);
   display.display();
}
```

Do programu z ćwiczenia 1 dołożono biblioteki i konfigurację dla wyświetlacza LED. Ponadto zamieniono stałą `KLUCZ` na zmienną typu całkowitego `int klucz`.

W programie klucz wczytywany jest z wejścia analogowego, do którego podłączony jest potencjometr. Instrukcja `klucz=analogRead(A1)/(1023/10);`. Po wczytaniu przeskalowanej wartości klucza zostaje ona wyświetlona na wyświetlaczu LED za pomocą instrukcji `seg7.writeGPIO(digit[klucz]);`.

6. Ćwiczenie 3 Do realizacji samodzielnej przez uczniów.

Napisz program, który odkoduje podstawieniem Cezara wiadomość przesłaną za pomocą transmisji UART. Odkodowany tekst zostanie wyświetlony na ekranie OLED.

Do realizacji ćwiczenia wczytaj klucz, który pobierzsz jako wskazanie wartości potencjometru. Przyjmij zakres wartości klucza od 0 do 9.

Wyświetl wartość klucza na wyświetlaczu LED.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED
#include "Adafruit_MCP23008.h"// biblioteka LED

int klucz;

string cezar;
// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);
```

```
// utworzenie obiektu wyświetlacza LED
Adafruit_MCP23008 seg7;
// utworzenie tablicy z cyframi dla wyświetlacza LED
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.setTextColor(WHITE);
 // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
   // konfiguracja pinów ekspandera wyświetlacza LED
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
}

void loop() {
  while(Serial.available() == 0)
  {
    klucz=analogRead(A1)/(1023/10);
    seg7.writeGPIO(digit[klucz]);
  }

  if(Serial.available() > 0)
  {
    cezar = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
  }
```

```
    for(int i=0; i<cezar.length(); i++)
    {
      // poniżej znajduje się sprawdzenie czy znalizowany znak to litera
      if(cezar[i]>='A' && cezar[i]<='Z')
      {
      // jeżeli litera po POMNIEJSZENIU  o wartość klucza jest większa niż 'Z'
      // to przesuń się na koniec alfabetu, czyli o 26 znaków do przodu
        if(cezar[i]-klucz<'A')
         cezar[i]+=26;

        cezar[i]-=klucz;
      }
    }
     display.clearDisplay();
     display.setTextColor(WHITE);
     display.setCursor(0,0);
     display.println(cezar);
     display.display();
}
```

W programie zastosowano podobny algorytm pracy jak w programie „szyfrującym" z ćwiczenia 2.
Można też powiedzieć, że jest to *ten sam* algorytm, tylko z wartością klucza przeciwną (modulo 26).

Tak jak w poprzednich ćwiczeniach, analizowane są pojedyncze znaki przesłanego łańcucha znaków. Jedyna różnica to odejmowanie wartości klucza tam, gdzie w poprzednim ćwiczeniu dodawaliśmy, i odpowiednia cykliczność tej operacji „26 do przodu" zamiast „26 do tyłu", jeżeli w wyniku dostaliśmy 0 lub liczbę ujemną.

7. Ćwiczenie 4 Do realizacji wraz z nauczycielem.

Napisz program, który za pomocą transmisji UART wczyta łańcuch znaków.

Następnie na wyświetlaczu LCD powinno pojawić sie menu z 3 opcjami do wyboru:

- Szyfruj
- Odszyfruj
- Wczytaj UART

Po wybraniu opcji szyfruj lub odszyfruj program powinien zaszyfrować lub odszyfrować tekst podany podczas transmisji UART z kluczem wczytanym z wartości potencjometru. Rezultat działania wybranej opcji powinien być wyświetlony na wyświetlaczu OLED.

Jeżeli użytkownik wybierze opcję „Wczytaj UART" będzie to oznaczało ponowne nasłuchiwanie portu szeregowego w celu odebrania danych do szyfrowania/odszyfrowania.

Wybór opcji powinien być wykonany za pomocą strzałek góra/dół stanowiących przyciski monostabilne. Zatwierdzenie wybranej opcji to przycisk środkowy.

Wczytany z potencjometru klucz wyświetl na wyświetlaczu LED. Możliwe wartości klucza przyjmij jako od 0 do 9.

Przykładowy ekran oczekiwania na tekst z komputera (przesłany transmisją UART)

Przykładowe ekrany na wyświetlaczu LCD z zaznaczeniem opcji wybranej w MENU:

Przykładowy widok podsumowania działania programu szyfrującego tekst „ARDUINO UNO” z kluczem o wartości 6.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// bibliteka OLED
#include "Adafruit_MCP23008.h"// biblioteka LED
#include <hd44780.h>// biblioteka LCD
#include <hd44780ioClass/hd44780_I2Cexp.h>// biblioteka LCD

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define SW_CENTER 8
#define SW_UP 6
#define SW_DOWN 5

int klucz;
int wybor=1;

string cezar;
// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

// utworzenie obiektu wyświetlacza LED
Adafruit_MCP23008 seg7;

// utworzenie tablicy z cyframi dla wyświetlacza LED
uint8_t digit[10] = {
 B00111111, // "0"
 B00000110, // "1"
 B01011011, // "2"
 B01001111, // "3"
```

```
 B01100110, // "4"
 B01101101, // "5"
 B01111101, // "6"
 B00000111, // "7"
 B01111111, // "8"
 B01101111, // "9"
};

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.setTextColor(WHITE);
 // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
   // konfiguracja pinów ekspandera wyświetlacza LED
 seg7.begin(0x4);
 seg7.pinMode(0, OUTPUT);
 seg7.pinMode(1, OUTPUT);
 seg7.pinMode(2, OUTPUT);
 seg7.pinMode(3, OUTPUT);
 seg7.pinMode(4, OUTPUT);
 seg7.pinMode(5, OUTPUT);
 seg7.pinMode(6, OUTPUT);
 seg7.pinMode(7, OUTPUT);
 // konfiguracja LCD
  lcd.begin(16, 2);
}

void menu()
{
  lcd.clear();
  if(wybor==1)
  {
    lcd.setCursor(0, 0);
    lcd.print("<Szyfruj>");
    lcd.setCursor(0, 1);
    lcd.print(" Odszyfruj");
  }
  if(wybor==2)
  {
    lcd.setCursor(0, 0);
    lcd.print(" Szyfruj");
    lcd.setCursor(0, 1);
    lcd.print("<Odszyfruj>");
  }
  if(wybor==3)
```

```
  {
    lcd.setCursor(0, 0);
    lcd.print(" Odszyfruj");
    lcd.setCursor(0, 1);
    lcd.print("<Wczytaj UART>");
  }
}

void czytajKlucz()
{
  klucz=analogRead(A1)/(1023/10);
  seg7.writeGPIO(digit[klucz]);
}
void wczytajUart()
{
  while(Serial.available() == 0)
  {
    czytajKlucz();
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Czekam na UART");
    delay(100);
  }
  if(Serial.available() > 0)
  {
    cezar = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
  }
}

void wyswietlOLED()
{
   display.clearDisplay();
   display.setTextColor(WHITE);
   display.setCursor(0,0);
   display.println(cezar);
   display.display();
}

void odszyfruj()
{
    for(int i=0; i<cezar.length(); i++)
  {
    // poniżej znajduje się sprawdzenie czy znalizowany znak to litera
    if(cezar[i]>='A' && cezar[i]<='Z')
    {
```

```
    // jeżeli litera po POMNIEJSZENIU  o wartość klucza jest większa niż 'Z'
    // to przesuń się na koniec alfabetu, czyli o 26 znaków do przodu
      if(cezar[i]-klucz<'A')
       cezar[i]+=26;
      cezar[i]-=klucz;
    }
  }
  wyswietlOLED();
}

void szyfruj()
{
  for(int i=0; i<cezar.length(); i++)
  {
    // poniżej znajduje się sprawdzenie czy znalizowany znak to litera
    if(cezar[i]>='A' && cezar[i]<='Z')
    {
    // jeżeli litera po zwiększeniu o wartość klucza jest większa niż 'Z'
    // to przesuń się do początku alfabetu, czyli o 26 znaków wstecz
      if(cezar[i]+klucz>'Z')
       cezar[i]-=26;

      cezar[i]+=klucz;
    }
  }
  wyswietlOLED();
}
void loop() {
  wczytajUart();

  while(digitalRead(SW_CENTER) == LOW)
  {
    if(digitalRead(SW_DOWN) == HIGH&&wybor<3)
    {
      wybor++;
    }
    if(digitalRead(SW_UP) == HIGH&&wybor>1)
    {
      wybor--;
    }
    menu();
    czytajKlucz();
    delay(200);
  }
  switch(wybor)
  {
    case 1:
```

```
        szyfruj();
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Wykonane :-)");
    delay(5000);

        break;
      case 2:
        odszyfruj();
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Wykonane :-)");
    delay(5000);

        break;
      case 3:

        break;
    }
  }
```

Realizację programu podzielono na kilka osobnych funkcji własnych użytkownika:

- `void czytajKlucz()` //funkcja czyta i skaluje do zakresu od 0 do 9 wartość potencjometru
- `void menu()` //funkcja wyświetlająca menu
- `void wczytajUart()` //funkcja oczekująca na łańcuch znaków UART
- `void wyswietlOLED()` //funkcja wyświetlająca wynik działania programu na wyświetlaczu OLED
- `void odszyfruj()` //funkcja odszyfrowująca przesłany łańcuch znakowy
- `void szyfruj()` //funkcja szyfrująca otrzymany łańcuch znakowy

Po konfiguracji wszystkich peryferiów w funkcji ... i po definicji wszystkich funkcji własnych użytkownika przechodzimy do funkcji ... .

W pierwej kolejności wczytujemy łańcuch znakowy z UART, a następnie oczekujemy na zatwierdzenie wybranej opcji z MENU za pomocą przycisku środkowego `SW_CENTER`. Oczekiwanie na przycisk wykonane jest za pomocą pętli ..., w której sprawdzamy, czy nie naciśnięto przycisków `SW_UP` lub `SW_DOWN`, dla których będziemy odpowiednio modyfikować zmienną pomocniczą `int wybor`, któ®a pamięta aktualną opcję MENU wybraną przez użytkownika.

Kiedy przycisk `SW_CENTER` zostaje naciśnięty, przechodzimy do funkcji `switch()`. Funkcja ta wypiera swoją dalszą ścieżkę realizacji w zależności od wartości podanej w parametrze. W naszym przypadku wartość przekazana w parametrze to zmienna `wybor`. Która przyjąć może 3 wartości. 1 dla opcji „Szyfruj", 2 dla opcji „Odszyfruj" i 3 dla opcji „Wczytaj UART". W zależności od wybranej opcji realizowany jest jeden z case'ów w funkcji `switch()`.

Na koniec następuje wyświetlenie wyników i po 5 sekundach program ponownie oczekuje na znaki przesłane po UART.

8. Przeprowadzenie quizu podsumowującego lekcję.

1. Słowo „INFORMATYKA” zakodowane podstawieniem Cezara z kluczem to:

```
0: INFORMATYKA     7: PUMVYTHAFRH    14: WBTCFAOHMYO    21: DIAJMHVOTFV
1: JOGPSNBUZLB     8: QVNWZUIBGSI    15: XCUDGBPINZP    22: EJBKNIWPUGW
2: KPHQTOCVAMC     9: RWOXAVJCHTJ    16: YDVEHCQJOAQ    23: FKCLOJXQVHX
3: LQIRUPDWBND    10: SXPYBWKDIUK    17: ZEWFIDRKPBR    24: GLDMPKYRWIY
4: MRJSVQEXCOE    11: TYQZCXLEJVL    18: AFXGJESLQCS    25: HMENQLZSXJZ
5: NSKTWRFYDPF    12: UZRADYMFKWM    19: BGYHKFTMRDT    26: INFORMATYKA
6: OTLUXSGZEQG    13: VASBEZNGLXN    20: CHZILGUNSEU
```

   a) PUMVYTHAFRH [klucz 7]
   b) QVNWZUIBGSI [klucz 8]
   c) RWOXAVJCHTJ [klucz 9]
   d) OTLUXSGZEQG [klucz 6]
2. Szyfrogram „DUGXLQR” po odszyfrowaniu szyfrem Cezara z kluczem 3 to:
   a) INFORMAT
   b) SZKOLENI
   c) ELEKTRO
   d) ARDUINO
3. Funkcję `switch()` możemy zastąpić:
   a) kilkoma funkcjami warunkowymi `if`
   b) pętlą `for()`
   c) pętlą `while()`
   d) jest niezastąpiona
4. Przy szyfrowaniu szyfrem Cezara
   a) Klucz nie jest konieczy do pracy z szyfrogramem
   b) Odbiorca musi posługiwać się tym samym kluczem
   c) Nadawca stosuje zawsze klucz o wartości 3
   d) Odbiorca stosuje zawsze klucz o wartości 6

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który zaszyfruje szyfrem Cezara tekst przesłany za pomocą transmisji UART. Zaszyfrowany tekst zostanie wyświetlony na ekranie OLED.

Do realizacji ćwiczenia zdefinuj w programie stałą KLUCZ. Testowo nadaj jej wartość 4.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>

#define KLUCZ 4

string cezar;

Adafruit_SSD1306 display(NULL);

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
 display.setTextColor(WHITE);
 // Ustawienie częstoliwości nadawania i odbioru
  Serial.begin(9600);
}

void loop() {
  while(Serial.available() == 0){}
  if(Serial.available() > 0)
  {
    cezar = Serial.readStringUntil('\n ');
    // przesłanie do komputera jednej linijki tekstu
    Serial.println("Odebrałem sygnal");
  }
  for(int i=0; i<cezar.length(); i++)
  {
// sprawdzamy, czy podany znak to litera:
if(cezar[i]>='A' && cezar[i]<='Z')

{
        // jeżeli litera po zwiększeniu o wartość klucza jest większa niż 'Z'
        // to przesuń się do początku alfabetu, czyli o 26 znaków wstecz
      if(cezar[i]+KLUCZ>'Z')
       cezar[i]-=26;
```

```
            cezar[i]+=KLUCZ;
          }
        }
         display.clearDisplay();
         display.setTextColor(WHITE);
         display.setCursor(0,0);
         display.println(cezar);
         display.display();
      }
```

Ćwiczenie 2

Napisz program, który zakoduje podstawieniem Cezara tekst przesłany za pomocą transmisji UART. Zakodowany tekst zostanie wyświetlony na ekranie OLED.

Do realizacji ćwiczenia wczytaj klucz, który pobierzsz jako wskazanie wartości potencjometru. Przyjmij zakres wartości klucza od 0 do 9.

Wyświetl wartość klucza na wyświetlaczu LED.

Dla dociekliwych: Po co zezwalamy na wartość klucza 0, jeżeli pozostawia on wiadomość niezaszyfrowaną (niezmienioną)? Wartość klucza 0 służy np. do sprawdzenia poprawności działania programu lub wyświetlacza.

Ćwiczenie 3

Napisz program, który odkoduje podstawieniem Cezara wiadomość przesłaną za pomocą transmisji UART. Odkodowany tekst zostanie wyświetlony na ekranie OLED.

Do realizacji ćwiczenia wczytaj klucz, który pobierzsz jako wskazanie wartości potencjometru. Przyjmij zakres wartości klucza od 0 do 9.

Wyświetl wartość klucza na wyświetlaczu LED.

Ćwiczenie 4

Napisz program, który za pomocą transmisji UART wczyta łańcuch znaków.

Następnie na wyświetlaczu LCD powinno pojawić sie menu z 3 opcjami do wyboru:

- Szyfruj
- Odszyfruj
- Wczytaj UART

Po wybraniu opcji szyfruj lub odszyfruj program powinien zaszyfrować lub odszyfrować tekst podany podczas transmisji UART z kluczem wczytanym z wartości potencjometru. Rezultat działania wybranej opcji powinien być wyświetlony na wyświetlaczu OLED.

Jeżeli użytkownik wybierze opcję „Wczytaj UART” będzie to oznaczało ponowne nasłuchiwanie portu szeregowego w celu odebrania danych do szyfrowania/odszyfrowania.

Wybór opcji powinien być wykonany za pomocą strzałek góra/dół stanowiących przyciski monostabilne. Zatwierdzenie wybranej opcji to przycisk środkowy.

Wczytany z potencjometru klucz wyświetl na wyświetlaczu LED. Możliwe wartości klucza przyjmij jako od 0 do 9.

Przykładowy ekran oczekiwania na tekst z komputera (przesłany transmisją UART)

Przykładowe ekrany na wyświetlaczu LCD z zaznaczeniem opcji wybranej w MENU:

Przykładowy widok podsumowania działania programu szyfrującego tekst „ARDUINO UNO” z kluczem o wartości 6.


1
VSSVDDVO RS RW E D0 D1 D2 D3 D4 D5 D6 D7 A K
16
Wykonane :-)
GXJAOTU ATU
LED1
LED2
LED3

Quiz

1. Słowo „INFORMATYKA” zakodowane podstawieniem Cezara z kluczem to:
   a) PUMVYTHAFRH
   b) QVNWZUIBGSI
   c) RWOXAVJCHTJ
   d) OTLUXSGZEQG
2. Szyfrogram „DUGXLQR” po odszyfrowaniu szyfrem Cezara z kluczem 3 to:
   a) INFORMAT
   b) SZKOLENI
   c) ELEKTRO
   d) ARDUINO
3. Funkcję `switch()` możemy zastąpić:
   a) kilkoma funkcjami warunkowymi `if`
   b) pętlą `for()`
   c) pętlą `while()`
   d) jest niezastąpiona
4. Przy szyfrowaniu szyfrem Cezara
   a) Klucz nie jest konieczy do pracy z szyfrogramem
   b) Odbiorca musi posługiwać się tym samym kluczem
   c) Nadawca stosuje zawsze klucz o wartości 3
   d) Odbiorca stosuje zawsze klucz o wartości 6

Lekcja 12

Inteligentny dom z zestawem edukacyjnym TME-EDU-ARD-2. – Cz. 1. Poznajemy czujnik temperatury i oświetlenia.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych i pętli w języku C++.
Utrwalenie umiejętności odczytu danych z potencjometru oraz przycisków monostabilnych w zestawie edukacyjnym TME-EDU-ARD-2.
Utrwalenie umiejętności pisania programów działających z wyświetlaczami LCD, graficznym OLED i LCD.
Zapoznanie z możliwościami odczytu i zastosowania danych pozyskanych z czujnika temperatury i czujnika natężenia światła.
Zapoznanie ucznia z założeniami „inteligentnego domu".

Efekty kształcenia

- Uczeń potrafi stosować zmienne w języku C++.
- Uczeń stosuje w swoich programach potencjometr oraz przyciski monostabilne do wprowadzania danych.
- Uczeń stosuje w swoich programach wyświetlacze LCD, graficzny OLED i LED.
- Uczeń w swoich programach używa danych odczytanych z czujnika temperatury.
- Uczeń w swoich programach używa danych odczytanych z czujnika natężenia światła.
- Uczeń zna założenia „inteligentnego domu"

Wstęp

Proponowana lekcja będzie stanowiła inspirację do przemyśleń na czym polega „Inteligentny dom" i jakie funkcjonalności może posiadać. Jest to jedna z dwóch lekcji, które przeprowadzą nas przez nowe komponenty zestawu edukacyjnego TME-EDU-ARD-2. W pierwszej części do podzespołów, które poznaliśmy już w innych lekcjach dołożymy czujnik temperatury i czujnik oświetlenia. Będziemy mogli zaprogramować reakcje na zmianę warunków oświetleniowo temperaturowych panujących w naszym otoczeniu.

Nasze programy będą symulacją reakcji na zaprogramowane zdarzenie. Jednak ich zmiana w przyszłości na programy wykonujące realne reakcje będzie bardzo prosta. I wymagać bę-

dzie jedynie podpięcia do wyprowadzeń Arduino UNO przekaźników i peryferii, które będą elementami wykonawczymi naszych programów.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

1. Jak programować poznane na poprzednich lekcjach podzespoły umożliwiające wprowadzanie danych, tj. Przyciski monostabilne, potencjometr.
2. Powtórzenie wiadomości o sposobie sterowania i wyświetlania danych na wyświetlaczu LCD, LED i wyświetlaczu graficznym OLED.

3. Omówienie przez nauczyciela specyfiki inteligentnego domu i możliwości symulacji zdarzeń w nim zachodzących.

Inteligentny dom to budynek, w którym zastosowano nowoczesne technologie. Nowoczesność inteligentnego domu to szerokie spektrum możliwych udoskonaleń mających na celu podwyższenie standardów użytkowania, ekologii budynku.

Inteligentny dom posiada całą sieć czujników sterowanych przez centralny układ zarządzający budynkiem. Centrala sterująca inteligentnym domem może o odpowiedniej porze dnia włączyć światło, zasłonić roletę, czy zaryglować drzwi. Może włączyć ogrzewanie określonej porze lub wtedy kiedy temperatura spadnie poniżej ustalonej. Alarmy, monitoring, światło, nawadnianie czy nawet dostawa artykułów pierwszej potrzeby to wszystko może wchodzić w skład komponentów inteligentnego domu.

W oparciu o zestaw edukacyjny TME-EDU-ARD-2 możemy zrealizować niemal wszystkie z nich, jednak potrzebowalibyśmy zewnętrznych komponentów stanowiących elementy wykonawcze. Takie jak: lampy, siłowniki, syreny, elektrozawory, piece, klimatyzatory. A także czujniki zewnętrzne światła, temperatury, ruchu, zamknięcia(stykowe) etc. Nie jesteśmy w stanie podczas lekcji zdobyć tych wszystkich akcesoriów, ale jesteśmy w stanie zaprogramować symulację tych wszystkich zdarzeń. Symulację opartą o wbudowane w zestaw edukacyjny czujniki i peryferii. A to będzie już pierwszym krokiem do późniejszej realizacji systemu sterowania inteligentnego domu z prawdziwego zdarzenia.

4. Omówienie przez nauczyciela sposobu działania czujnika temperatury wraz z instrukcjami odpowiedzialnymi za jego użycie.

W skład zestawu edukacyjnego wchodzi wbudowany czujnik temperatury MCP9701. Jest to czujnik, który w zależności od intensywności światła zmienia poziom napięcia na wejściu analogowym A2 układu ArduinoUNO.

Odczyt temperatury polega na odczycie danych z wejścia analogowego A2 i na ich dalszym przeskalowaniu. By poprawnie odczytać dane z tego czujnika w skali stopni Celsjusza, należy wartość odczytaną z czujnika poddać do wzoru:

```
Temp_°C = (dane_z_czujnika_temp)*0,125-22,0;
```

Do obsługi tego podzespołu nie potrzebujemy żadnych nowych bibliotek, ani instrukcji.

Przykładowa instrukcja odczytująca dane z czujnika i zapamiętująca je w zmiennej typu całkowitego to:

```
int odczyt = analogRead(A2);
int tempC = (int)(odczyt*0.125 -22.0);
```

Dokładność pomiaru temperatury przez czujnik to –2°C / +2°C.

Maksymalna temperatura pomiaru to 125°C.

5. Ćwiczenie 1 Do realizacji wraz z nauczycielem.

Napisz program, który będzie mierzył temperaturę w pomieszczeniu i wyświetlał ją na ekranie LCD.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

int odczyt = analogRead(A2);
int tempC;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  // pobranie odczytu temperatury
  odczyt = analogRead(A2);
  // przeskalowanie odczytu do wartości temp. w stopniach Celsjusza
  tempC = (int)(odczyt*0.125 - 22.0);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temperatura:");
  lcd.setCursor(0, 1);
  lcd.print(tempC);
  lcd.setCursor(4, 1);
  lcd.print("C");
  delay(200);
}
```

W programie odczyt temperatury w °C został wykonany w dwóch krokach. Najpierw ze zmiennej `odczyt` pobraliśmy dane z wejścia analogowego A2, do którego podłączony jest termometr. Następnie po przeskalowaniu przypisaliśmy wartość temperatury do zmiennej tempC.

W kolejnym kroku nastąpiło wyświetlenie wyniku na ekranie wyświetlacza LCD.

6. Ćwiczenie 2 Do realizacji samodzielnej przez uczniów.

Napisz program, który będzie mierzył temperaturę w pomieszczeniu i wyświetlał ją na ekranie LCD. Jeśli temperatura przekroczy 25°C, włączona zostanie dioda LED D1.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

int odczyt = analogRead(A2);
int tempC;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
  // Ustawienie pinu diody LED jako wyjście
  pinMode(13, OUTPUT);
}

void loop() {
  // pobranie odczytu temperatury
  odczyt = analogRead(A2);
  // przeskalowanie odczytu do wartości temp. w stopniach Celsjusza
  tempC = (int)(odczyt*0.125 - 22.0);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temperatura:");
  lcd.setCursor(0, 1);
  lcd.print(tempC);
  lcd.setCursor(4, 1);
  lcd.print("C");
  if(tempC>=25)
  {
    // włączenie diody
    digitalWrite(13, HIGH);
  }
  else
  {
    // wyłączenie diody
    digitalWrite(13, LOW);
  }
  delay(200);
}
```

Podobnie jak w programie z ćwiczenia 1, odczyt został przeskalowany do wartości w °C.

Po tej operacji w instrukcji warunkowej sprawdziliśmy, czy założona wartość 25°C została przekroczona. Jeśli warunek został spełniony, włączona została dioda LED D1.

Po obniżeniu temperatury instrukcja warunkowa wejdzie w blok instrukcji alternatywnych, czyli wyłączy diodę D1 na pinie 13.

7. Dyskusja w grupie z uczniami na temat:

W jaki sposób moglibyśmy zastosować umiejętność programowania pomiaru temperatury i wysterowania poszczególnych wyjść układu ArduinoUNO w rozwiązaniach związanych z „inteligentnym domem”?

W założeniach do dyskusji przyjmijmy, że możemy podłączać do układu Ardunino UNO dodatkowe urządzenia wykonawcze (grzałki, oświetlenie, elektrozawory etc.).

8. Ćwiczenie 3 Do realizacji samodzielnej przez uczniów.

Zmodyfikuj programy z ćwiczeń 1 i 2 tak, aby dioda LED RGB odzwierciedlała tonacją barwną temperaturę zmierzoną przez czujnik. Skala kolorów powinna uwzględniać zakres od 0 do 30°C. Dla 0°C powinniśmy wyświetlić kolor niebieski, dla 30°C kolor czerwony. Dla wartości środkowej, czyli 15°C, kolor powinien być zielony.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

#define LED2RED 9
#define LED2GREEN 10
#define LED2BLUE 11

int odczyt = analogRead(A2);
float tempC;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
  // Ustawienie pinu diody LED jako wyjście
  pinMode(13, OUTPUT);

  // konfiguracja pinów diody RGB jako pinów wyjściowych
 pinMode(LED2RED, OUTPUT);
 pinMode(LED2GREEN, OUTPUT);
 pinMode(LED2BLUE, OUTPUT);
}
uint8_t red;
uint8_t green;
uint8_t blue;

// funkcja przeliczająca wartość temp. na barwy RGB
// Wartości zostały dobrane doświadczalnie
// Proporcjonalne zmiany barw zachodzą w zakresie od 0 do 30°C
// Poniżej temp 0°C pozostaje kolor niebieski
```

```
// Powyżej temp 30°C pozostaje kolor czerwony
void tempToRGB(float temp)
{
  if(temp<7.5)
  {
    blue=255;
    green=255-((7.5-temp)/7.5)*255;
  }
  else
    if(temp<15)
    {
      green=255;
      blue=255-((temp-7.5)/7.5)*255;
    }
    else
        if(temp<22.5)
        {
          green=255;
          red=255-((22.5-temp)/7.5)*255;
        }
        else
        {
          red=255;
          green=255-((temp-22.5)/7.5)*255;
        }
}

void loop() {
  // pobranie odczytu temperatury
  odczyt = analogRead(A2);
  // przeskalowanie odczytu do wartości temp. w stopniach Celsjusza
  tempC = (float)(odczyt*0.125 - 22.0);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temperatura:");
  lcd.setCursor(0, 1);
  lcd.print(tempC);
  lcd.setCursor(4, 1);
  lcd.print("C");
  tempToRGB(tempC);
  analogWrite(LED2RED, red);
  analogWrite(LED2GREEN, green);
  analogWrite(LED2BLUE, blue);
  delay(200);
}
```

W programie zastosowano trzy zmienne `red`, `green` i `blue` typu `uint8_t`. Jest to typ całkowity o zakresie od 0 do 255. Zmienne te posłużą nam do określenia wartości intensywności poszczególnych barw na diodzie świecącej RGB.

Po odczytaniu wartości temperatury z czujnika wywołana zostanie funkcja własna użytkownika `tempToRGB(tempC)`, której zadaniem jest określenie wartości parametrów diody RGB.

Funkcja `void tempToRGB(float temp)` przedstawia serię zagnieżdżonych warunków, które w widełkach co 7,5°C przeliczają zmiany poszczególnych kolorów czerwonego, zielonego i niebieskiego w diodzie RGB. Zmiany barw zostały określone doświadczalnie i dla zera stopni i poniżej dają barwę niebieską. W środku skali mamy barwy zieleni, a przy 30 i więcej stopniach Celsjusza – kolor czerwony.

W ostatnim kroku następuje wysterowanie wyjść analogowych układu ArduinoUNO tak, by przesłały wartości poszczególnych kolorów.

9. Omówienie przez nauczyciela sposobu działania czujnika intensywności światła wraz z instrukcjami odpowiedzialnymi za jego użycie.

Czujnik intensywności światła dołączony do zestawu edukacyjnego działa podobnie jak czujnik temperatury. Tzn., zmiana intensywności światła powoduje zmianę napięcia na wejściu analogowym A3. Dołączony sensor światła to KPS-3227, a odczyt z niego możemy robić w 10-bitowej skali. Dla maksymalnego zaciemnienia wartość odczytana z wejścia analogowego jest równa 0 (zero). Im jest jaśniej, tym wyższe wskazanie czujnika, aż do max. 1023.

Przykładowa instrukcja odczytująca poziom intensywności oświetlenia w skali od 0 do 1023 i zapamiętująca pomiar w zmiennej całkowitej `jasnosc` :

```
int jasnosc = analogRead(A3);
```

10. Ćwiczenie 4 Do realizacji wraz z nauczycielem.

Napisz program, który będzie mierzył intensywność światła w pomieszczeniu i wyświetlał ją na ekranie LCD w skali od 0 do 100.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// deklaracja zmiennej przechowującej pomiar oświetlenia
int jasnosc = analogRead(A3);
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  // pobieramy pomiar z czujnika oświetlenia
  jasnosc = analogRead(A3);
  // skalujemy pomiar w zakresie od 0 do 100
```

```
    jasnosc=jasnosc/(10.23);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Oswietlenie:");
    lcd.setCursor(0, 1);
    lcd.print(jasnosc);
    delay(200);
  }
```

W programie za pomocą polecenia odczytu z wejścia analogowego `analogRead(A3)` odczytano dane z czujnika oświetlenia. Zostały one zapisane w zmiennej `jasnosc`, której wartość została następnie przeskalowana do zakresu od 0 do 100.

11. Ćwiczenie 5 Do realizacji samodzielnej przez uczniów

Napisz program, który do programu z ćwiczenia 4 będzie miał dołożony próg wartości intensywności światła, poniżej którego załączy się dioda LED D13.

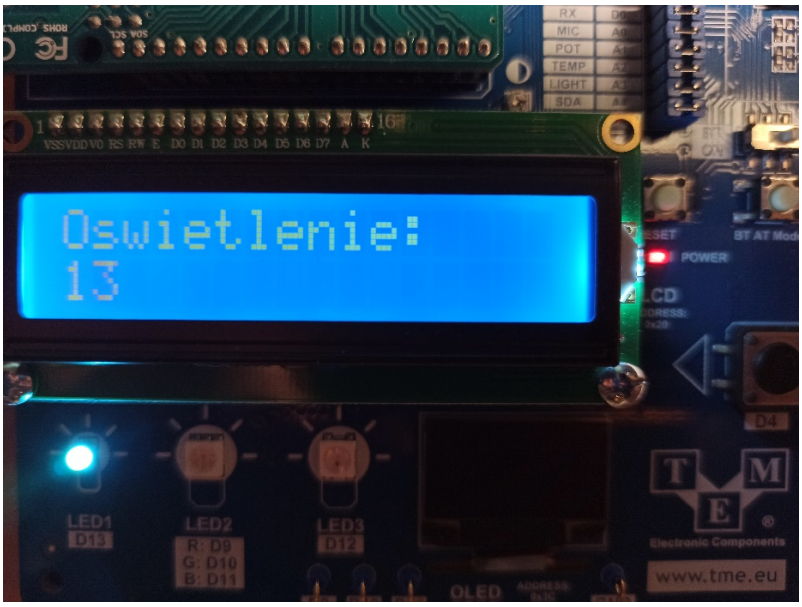
Próg powinien posiadać możliwość regulacji potencjometrem.

Wykonaj program tak, by wejście w konfigurację progu i jego zatwierdzenie następowało po naciśnięciu przycisku monostabilnego `SW_CENTER`.

W takcie konfiguracji progu powinna się wyświetlać na wyświetlaczu LCD wartość tego progu.

Przykładowy widok wyświetlacza LCD, kiedy znajdziemy się w trybie konfiguracji progu:

Przykładowy widok standardowej pracy i pomiaru oświetlenia (wartość oświetlenia spadła poniżej progu – dioda LED D1 jest włączona):



Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);
#define SW_CENTER 8

// deklaracja zmiennej przechowującej pomiar oświetlenia
int jasnosc = analogRead(A3);
int prog=0;// zmienna przechowująca wartość ustawionego progu

// zmienna stanowiąca flagę wejścia w tryb konfiguracji
bool konfigProgu=0;

void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);

  // Ustawienie pinu diody LED jako wyjście
  pinMode(13, OUTPUT);
}

void loop() {
  // instrukcja warunkowa w której sprawdzamy naciśnięcie przycisku środkowego
  if (digitalRead(SW_CENTER) == HIGH)
  {
    // zmieniamy stan flagi wejścia w konfigurację progu na stan przeciwny
     konfigProgu=!konfigProgu;
     delay(500);
  }
```

```
  // sprawdzamy czy jesteśmu w trybie konfiguracji progu
  if(konfigProgu==1)
  {
    prog=analogRead(A1)/(10.23);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Prog oswietlenia:");
    lcd.setCursor(0, 1);
    lcd.print(prog);
  }
  else
  {
    // pobieramy pomiar z czujnika oświetlenia
    jasnosc = analogRead(A3);
    // skalujemy pomiar w zakresie od 0 do 100
    jasnosc=jasnosc/(10.23);
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Oswietlenie:");
    lcd.setCursor(0, 1);
    lcd.print(jasnosc);

    // instrukcja warunkowa sprawdzająca,
    // czy intensywnosc oświetlenia nie jest mniejsza niż ustawiony próg
    if(prog>=jasnosc)
      digitalWrite(13, HIGH);
    else
      digitalWrite(13, LOW);

  }
  delay(200);
}
```

W programie zdefiniowano zmienną całkowitą `prog`, która będzie przechowywała wartość progu, przy którym nastąpi włączenie diody LED. Zastosowano ponadto jeszcze jedną nową zmienną – typu logicznego: `konfigProgu`. Zmienna `konfigProgu` stanowi w programie flagę, która określa, czy program znajduje się w trybie konfiguracji progu załączania diody LED.

Jeśli przycisk SW_CENTER jest wciśnięty to flaga konfigProgu zostaje podniesiona (ma wartość 1).

Ten stan zostaje wykryty przez kolejną instrukcję warunkową, która odczytuje dane z wejścia analogowego A1, pod które podłączony jest potencjometr. Odczytana wartość pobrana zostanie do zmiennej `prog` i wyświetlona na ekranie LCD.

Przy ponownym wciśnięciu przycisku SW_CENTER wartość flagi zostanie ustawiona na przeciwną. Czyli w opisywanej sytuacji zostanie opuszczona.

— Dyskusja w grupie z uczniami na temat:

W jaki sposób moglibyśmy zastosować umiejętność programowania pomiaru intensywności światła i wysteorowania poszczególnych wyjść układu ArduinoUNO w rozwiązaniach związanych z „inteligentnym domem”?

W założeniach do dyskusji przyjmijmy, że możemy podłączać do układu Ardunino UNO dodatkowe urządzenia wykonawcze(grzałki, oświetlenie, elektrozawory, siłowniki rolet, wentylatory etc.).

12. Przeprowadzenie quizu podsumowującego lekcję.

1. Czujnik temperatury został dołączony w zestawie edukacyjnym TME-EDU-ARD-2 do układu ArduinoUNO do wejścia:
   a) A0
   b) A1
   c) A2
   d) A3
2. Odczytana wartość wejścia analogowego Arduino UNO z podłączonym czujnikiem temperatury jest:
   a) od razu podana w °C
   b) od razu podana w °F
   c) podana w skali od 0 do 100
   d) podana w skali od 0 do 1023
3. Zaznacz, czego nie zakwalifikujemy do znamion „inteligentnego domu”:
   a) Czujnika zmierzchowego świateł zewnętrznych.
   b) Termostatycznie sterowanego ogrzewania.
   c) Telewizora.
   d) Alarmu z powiadamianiem GSM.
4. Odczytaną wartość temperatury możemy określić z dokładnością:
   a) 0.1°C
   b) 0.5°C
   c) 1°C
   d) 2°C
5. Maksymalna temperatura pomiaru czujnika MCP9701 to:
   a) 100°C
   b) 125°C
   c) 250°C
   d) 373 K

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który będzie mierzył temperaturę w pomieszczeniu i wyświetlał ją na ekranie LCD.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

int odczyt = analogRead(A2);
int tempC;
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  // pobranie odczytu temperatury
  odczyt = analogRead(A2);
  // przeskalowanie odczytu do wartości temp. w stopniach Celsjusza
  tempC = (int)(odczyt*0.125 - 22.0);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temperatura:");
  lcd.setCursor(0, 1);
  lcd.print(tempC);
  lcd.setCursor(4, 1);
  lcd.print("C");
  delay(200);
}
```

Ćwiczenie 2

Napisz program, który będzie mierzył temperaturę w pomieszczeniu i wyświetlał ją na ekranie LCD. Jeśli temperatura przekroczy 25°C, włączona zostanie dioda LED D1.

Ćwiczenie 3

Zmodyfikuj programy z ćwiczeń 1 i 2 tak, aby dioda LED RGB odzwierciedlała tonacją barwną temperaturę zmierzoną przez czujnik. Skala kolorów powinna uwzględniać zakres od 0 do 30°C. Dla 0°C powinniśmy wyświetlić kolor niebieski, dla 30°C kolor czerwony. Dla wartości środkowej, czyli 15°C, kolor powinien być zielony.

Ćwiczenie 4

Napisz program, który będzie mierzył intensywność światła w pomieszczeniu i wyświetlał ją na ekranie LCD w skali od 0 do 100.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// deklaracja zmiennej przechowującej pomiar oświetlenia
int jasnosc = analogRead(A3);
void setup(){
  // konfiguracja LCD
  lcd.begin(16, 2);
}

void loop() {
  // pobieramy pomiar z czujnika oświetlenia
  jasnosc = analogRead(A3);
  // skalujemy pomiar w zakresie od 0 do 100
  jasnosc=jasnosc/(10.23);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Oswietlenie:");
  lcd.setCursor(0, 1);
  lcd.print(jasnosc);
  delay(200);
}
```

Ćwiczenie 5

Napisz program, który do programu z ćwiczenia 4 będzie miał dołożony próg wartości intensywności światła, poniżej którego załączy się dioda LED D13.

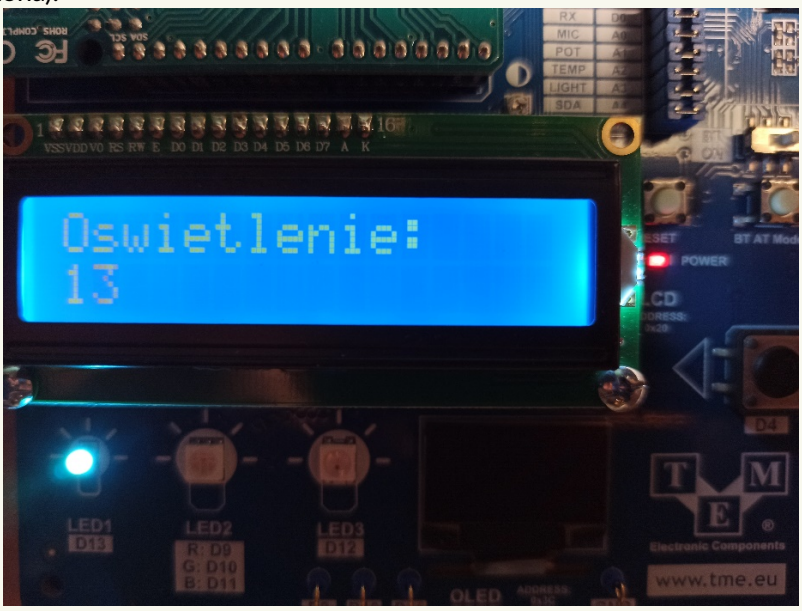
Próg powinien posiadać możliwość regulacji potencjometrem.

Wykonaj program tak, by wejście w konfigurację progu i jego zatwierdzenie następowało po naciśnięciu przycisku monostabilnego `SW_CENTER`.

W takcie konfiguracji progu powinna się wyświetlać na wyświetlaczu LCD wartość tego progu.

Przykładowy widok wyświetlacza LCD, kiedy znajdziemy się w trybie konfiguracji progu:

Przykładowy widok standardowej pracy i pomiaru oświetlenia (wartość oświetlenia spadła poniżej progu – dioda LED D1 jest włączona):



Quiz

1. Czujnik temperatury został dołączony w zestawie edukacyjnym TME-EDU-ARD-2 do układu ArduinoUNO do wejścia:
   a) A0
   b) A1
   c) A2
   d) A3
2. Odczytana wartość wejścia analogowego Arduino UNO z podłączonym czujnikiem temperatury jest:
   a) od razu podana w °C
   b) od razu podana w °F
   c) podana w skali od 0 do 100
   d) podana w skali od 0 do 1023
3. Zaznacz, czego nie zakwalifikujemy do znamion „inteligentnego domu”:
   a) Czujnika zmierzchowego świateł zewnętrznych.
   b) Termostatycznie sterowanego ogrzewania.
   c) Telewizora.
   d) Alarmu z powiadamianiem GSM.
4. Odczytaną wartość temperatury możemy określić z dokładnością:
   a) 0.1°C
   b) 0.5°C
   c) 1°C
   d) 2°C
5. Maksymalna temperatura pomiaru czujnika MCP9701 to:
   a) 100°C
   b) 125°C
   c) 250°C
   d) 373 K

Lekcja 13

Inteligentny dom z zestawem edukacyjnym TME-EDU-ARD-2. – Cz. 2. Poznajemy działanie czujnika dźwięku. Problem wielozadaniowości w projektach z układem ArduinoUNO. Zapamiętujemy dane konfiguracyjne w pamięci nieulotnej EEPROM.

Czas trwania lekcji: **90 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych oraz pętli w języku C++. Utrwalenie umiejętności odczytu danych z potencjometru, przycisków monostabilnych, czujnika temperatury i czujnika natężenia światła w zestawie edukacyjnym TME-EDU-ARD-2.
Utrwalenie umiejętności pisania programów działających z wyświetlaczami LCD, graficznym OLED i LCD.
Zapoznanie z możliwościami odczytu i zastosowania danych pozyskanych z mikrofonu.
Zapoznanie z rodzajami pamięci. SRAM, EEPROM i FLASH.
Zapoznanie z problemem wielozadoniowości w pracy mikrokontrolera.

Efekty kształcenia

- Uczeń potrafi stosować zmienne w języku C++.
- Uczeń stosuje w swoich programach potencjometr oraz przyciski monostabilne do wprowadzania danych.
- Uczeń stosuje w swoich programach wyświetlacze LCD, graficzny OLED i LED.
- Uczeń w swoich programach używa danych odczytanych z czujnika temperatury.
- Uczeń w swoich programach używa danych odczytanych z czujnika natężenia światła.
- Uczeń w swoich programach używa danych odczytanych z czujnika mikrofonu.
- Uczeń rozróżnia pamięci SRAM, EEPROM i FLASH.
- Uczeń stosuje pamięć EEPROM do zapamiętywania danych konfiguracyjnych układu.
- Uczeń wie na czym polega wielozadaniowość mikrokontrolera

Wstęp

Proponowana lekcja to kontynuacja lekcji, która stanowiła inspirację do przemyśleń na czym polega „inteligentny dom" i jakie funkcjonalności może posiadać. W pierwszej części poznaliśmy czujnik temperatury i czujnik oświetlenia, który mógł po podłączeniu komponentów zewnętrznych załączać ogrzewanie lub klimatyzację, a także sterować oświetleniem jako włącznik zmierzchowy.

Ta lekcja będzie okazją do poznania działania i zastosowania w naszych programach czujnika dźwięku. Dodatkowo uświadomimy sobie, że nasze programy nie mogą w swoich zmiennych bazować tylko na pamięci ulotnej SRAM. Dowiemy się, jak możemy tę pamięć zastąpić w naszych projektach, by dane w niej zapisywane były przechowywane do kolejnego uruchomienia układu.

Tak jak w poprzedniej lekcji, część projektowa będzie stanowiła symulację reakcji na zaprogramowane zdarzenie realizujące założenia „inteligentnego domu".

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

–
  - Jak programować poznane na poprzednich lekcjach podzespoły umożliwiające wprowadzanie danych, tj. Przyciski monostabilne, potencjometr, czujnik temperatury, czujnik światła.
  - Powtórzenie wiadomości o sposobie sterowania i wyświetlania danych na wyświetlaczu LCD, LED i wyświetlaczu graficznym OLED.

3. Omówienie przez nauczyciela sposobu działania czujnika dźwięku wraz z instrukcjami odpowiedzialnymi za jego użycie.

Mikrofon, który został wbudowany w zestaw edukacyjny TME-EDU-ARD-2 jest to prosty czujnik dźwięku. Pełni on funkcję czujnika poziomu natężenia dźwięku i nie uda nam się go wykorzystać to projektów w rodzaju „rejestrator dźwięku" czy „dyktafon".

Do jego wykorzystania nie potrzebujemy żadnych nowych bibliotek.

Sam mikrofon podpięty jest do wejścia analogowego A0 w układzie Arduino za pomocą dodatkowych układów wzmacniających i stabilizujących dźwięk „łapany" z otoczenia.

Do pracy z czujnikiem wykorzystamy polecenie `analogRead(A0)`. Instrukcja ta podobnie jak w czujniku temperatury, dźwięku czy odczycie wartości z potencjometru będzie odczytywała z mikrofonu poziom natężenia głosu w skali 10 bitowej tj. od 0 do 1023.

4. Ćwiczenie 1 Do realizacji wraz z nauczycielem.

Napisz program, który odczyta wartość z czujnika dźwięku i wyświetli poziom głośności na wyświetlaczu OLED.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED

// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
}

void loop() {
   display.clearDisplay();
   display.setTextColor(WHITE);
   display.setCursor(0,0);
   display.println("Poziom halasu");
   display.setCursor(0,7);
   display.println("wynosi");
   display.setCursor(0,14);
   // wyświetlenie wartości odczytanej z mikrofonu.
   display.println(analogRead(A0));
   display.display();
   delay(200);
}
```

W programie wartość intensywności dźwięku w otoczeniu zostanie odczytana z wejścia analogowego A0 za poleceniem `analogRead(A0)`, i od razu wysłana na wyświetlacz OLED.

Wartość ta będzie zatem liczbą z zakresu od 0 do 1023.

5. Ćwiczenie 2 Do realizacji samodzielnej przez uczniów

Napisz program, który odczyta wartość z czujnika dźwięku i jeśli poziom hałasu przekroczy wartość 500, to włączona zostanie dioda led D1 na 5 sekund.

Oceń, czy odczytany z układu poziom hałasu o wartości 500 może być już uznany za głośny i dokuczliwy.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED

// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);
   // Ustawienie pinu diody LED jako wyjście
  pinMode(13, OUTPUT);
}

void loop() {
```

```
    display.clearDisplay();
    display.setTextColor(WHITE);
    display.setCursor(0,0);
    display.println("Poziom halasu");
    display.setCursor(0,7);
    display.println("wynosi");
    display.setCursor(0,14);
    // wyświetlenie wartości odczytanej z mikrofonu.
    display.println(analogRead(A0));
    display.display();
     // instrukcja warunkowa sprawdzająca,
     // czy hałas nie jest większy niż 500
     if(500<=analogRead(A0))
     {
       digitalWrite(13, HIGH);
       delay(5000);
     }
    digitalWrite(13, LOW);
    delay(200);
}
```

W programie po odczytaniu wartości intensywności dźwięku sprawdzany jest jego poziom. Instrukcja warunkowa sprawdza, czy przekroczona została wartość 500. Jeśli tak, to włączana jest kontrolka „hałasu”. W tym ćwiczeniu wskaźnikiem „hałasu” jest dioda LED D1.

6. Ćwiczenie 3 Do realizacji samodzielnej przez uczniów

Napisz program, w którym zmodyfikujesz projekt z ćwiczenia 2 tak, by próg załączania diody LED D1 był konfigurowany przez przyciski monostabilne „góra”/„dół”. Wartość progu załączania diody D1 powinna być wyświetlana na wyświetlaczu LCD, a poziom hałasu na wyświetlaczu OLED.

Skróć czas świecenia diody LED D1 do 1 sekundy po wykryciu przekroczenia progu.

Odpowiedz w grupie na poniższe pytania:

- Jakie zastosowanie może mieć czujnik poziomu dźwięku w centrali sterowania inteligentnym domem?
- Jakie niedoskonałości w pomiarach natężenia dźwięku widzimy?
- Czy mierzenie chwilowej wartości dźwięku jest dobrym rozwiązaniem?
- Jakie możesz zaproponować inne możliwości pomiaru i analizy poziomu dźwięku (hałasu)? Np. średnia z 10 pomiarów wykonanych co 10 ms. Podaj wady i zalety tych rozwiązań.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
```

```
// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

#define SW_UP 6
#define SW_DOWN 5

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);
  // Ustawienie pinu diody LED jako wyjście
  pinMode(13, OUTPUT);

  // konfiguracja LCD
  lcd.begin(16, 2);
}

int prog=0;
void loop() {
  if(digitalRead(SW_DOWN) == HIGH&&prog>1)
  {
    prog--;
  }
  if(digitalRead(SW_UP) == HIGH&&prog<1023)
  {
    prog++;
  }
  // poniżej wyświetlam wartość progu
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Prog halasu::");
  lcd.setCursor(0, 1);
  lcd.print(prog);

  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Poziom halasu");
  display.setCursor(0,7);
  display.println("wynosi");
  display.setCursor(0,14);
  // wyświetlenie wartości odczytanej z mikrofonu.
  display.println(analogRead(A0));
  display.display();
  // instrukcja warunkowa sprawdzająca,
  // czy hałasu nie jest większa niż wartość progu
```

```
    if(prog<=analogRead(A0))
    {
      digitalWrite(13, HIGH);
      delay(1000);
    }
    digitalWrite(13, LOW);
    delay(50);
  }
```

W programie zostały wprowadzone instrukcje warunkowe, które kontrolują stan przycisków „góra”/„dół”. Jeśli wykryte zostanie przyciśnięcie przycisku `SW_DOWN` („dół”), to wartość zmiennej `prog` zostaje zmniejszona o 1. Przyciśnięcie przycisku `SW_UP` („góra”) to nastąpi zwiększenie wartości progu o 1.

W kolejnej instrukcji warunkowej sprawdzamy, czy wartość progu została przekroczona. Jeśli tak, to włączona zostaje dioda LED D1.

7. Omówienie przez nauczyciela problemu wielozadaniowości w projektach z Arduino UNO.

W programach, które do tej pory tworzyliśmy dla dodania opóźnień np. między kolejnymi odczytami lub załączaniem/wyłączeniem diody LED używaliśmy funkcji `delay()`. Funkcja ta wstrzymywała program na określoną ilość milisekund. Powodowało to, że nasz program nie mógł wykonać w tym czasie innych czynności.

W aktualnych ćwiczeniach zauważyliśmy, że jeśli poziom hałasu przekroczy pewien próg, a dioda LED włączy się na ustaloną ilość sekund, to nie możemy wprowadzać wtedy zmian w wartości progu za pomocą przycisków monostabilnych.

A co, jeśli układ centrali „inteligentnego domu” miałby np. oznajmiać stan gotowości mruganiem diody LED? Nie moglibyśmy w tym czasie (mrugania diody) dokonywać pomiarów z czujników temperatury, oświetlenia, dźwięku etc.

Na szczęście jest pewna alternatywa dla funkcji `delay()`. To funkcja `millis()`, która zwraca liczbę milisekund, które upłynęły od włączenia zasilania bądź resetu programu.

Jak możemy wykorzystać funkcję `millis()` w naszych programach? Nic prostszego. Wystarczy, że będziemy pamiętać czas np. załączania diody LED. Czas załączenia diody pobierzemy do zmiennej całkowitej właśnie funkcją `millis()`. W kolejnych realizacjach pętli funkcji `loop()` będziemy sprawdzać jaka jest różnica czasu między aktualnym odczytem, a wartością czasu zapamiętaną w momencie załączenia diody LED. Nasza dioda będzie wykonywała rekcję wyłączenia właśnie w chwili, gdy różnica między czasem aktualnym i pobranym na starcie jest większa niż założona.

Takie podejście powoduje, że możemy wykonywać wiele czynności jednocześnie. Ewentualne reakcje będziemy wykonywać w zależności od zmierzonego czasu. Projekty wykonujące jednocześnie więcej niż jeden proces będziemy nazywać **wielozadaniowymi**.

wielozadaniowość

8. Ćwiczenie 4 Do realizacji wraz z nauczycielem

Napisz program, w którym zmodyfikujesz projekt z ćwiczenia 3 tak, by zmiana progu nie była opóźniona funkcją `delay()`.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

#define SW_UP 6
#define SW_DOWN 5

// zmienna przechowująca odczyt czasu od włączenia układu
unsigned long aktCzas = 0;

// zmienna przechowuje moment włączenia diody LED
unsigned long czasLED1_on = 0;

// zmienna pomocnicza do obliczenia czasu, który upłynął od włączenia diody
unsigned long roznicaCzasu = 0;

// zmienna ”flaga” podnoszona w chwili włączenia diody LED
bool LED1_on=0;

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);
  // Ustawienie pinu diody LED jako wyjście
  pinMode(13, OUTPUT);

  // konfiguracja LCD
  lcd.begin(16, 2);
}

int prog=0;
void loop() {

  if(digitalRead(SW_DOWN) == HIGH&&prog>1)
  {
    prog--;
  }
  if(digitalRead(SW_UP) == HIGH&&prog<1023)
  {
    prog++;
  }
  // poniżej wyświetlam wartość progu
  lcd.clear();
```

```
    lcd.setCursor(0, 0);
    lcd.print("Prog halasu:");
    lcd.setCursor(0, 1);
    lcd.print(prog);

    display.clearDisplay();
    display.setTextColor(WHITE);
    display.setCursor(0,0);
    display.println("Poziom halasu");
    display.setCursor(0,7);
    display.println("wynosi");
    display.setCursor(0,14);
    // wyświetlenie wartości odczytanej z mikrofonu.
    display.println(analogRead(A0));

    display.setCursor(0,21);
    display.println(millis());
    display.display();
    // instrukcja warunkowa sprawdzająca,
    // czy hałasu nie jest większa niż wartość progu
    // wtedy podnoszona jest ”flaga” włączenia diody LED
    // i przypisywany jest czas w kótrym to włączenie nastąpiło

    if(prog<=analogRead(A0))
    {
      LED1_on=1;
      czasLED1_on = millis();
    }

    // jeśli ”flaga” włączenia diody jest podniesiona
    // i różnica zaczy od włączenia jest mniejsza niż 1 sekunda
    // to włącz diodę LED D1 i zmierz czas
    // w przeciwnym wypadku wyłącz diodę LED, opuść ”flagę” i wyzeruj licznik czasu
    if(LED1_on==1 && roznicaCzasu<1000)
    {
      // Pobierz czas, który minął od staru programu
      aktCzas = millis();
      roznicaCzasu = aktCzas - czasLED1_on;
      digitalWrite(13, HIGH);
    }
    else
    {
      digitalWrite(13, LOW);
      LED1_on=0;
      roznicaCzasu=0;
    }
  }
```

W programie zadeklarowano trzy dodatkowe zmienne typu całkowitego `aktCzas`, `czasLED1_on`, `roznicaCzasu`. Zmienne te będą nam pomocne do odliczania czasu, który upłynął od momentu np. włączenia diody LED.

Ponadto wprowadzono zmienną logiczną `LED1_on` stanowiącą flagę, która sygnalizuje włączenie diody LED D1.

W programie w chwili wykrycia przez instrukcję warunkową przekroczenia progu hałasu podnoszona jest flaga `LED1_on` oraz zapisywany jest czas tego przekroczenia (podniesienia flagi).

Kolejna instrukcja warunkowa sprawdza, czy flaga `LED1_on` jest podniesiona i czy roznicaCzasu od włączenia diody nie przekroczyła 1000 ms. Jeśli tak, to włączana jest dioda LED D1 i obliczana jest różnica czasu jaka upłynęła od momentu włączenia.

Jeśli `roznicaCzasu` wynosi więcej niż 1 sekundę, to opuszczana jest flaga `LED1_on` oraz zerowany jest licznik czasu tj. zmienna `roznicaCzasu`.

Takie rozwiązanie powoduje, że cały czas płynnie mikrokontroler wykonuje wszystkie zadania. Nie ma „sztucznych” przestoi np. po włączeniu diody LED. Program nie czeka, aż dioda się wyłączy, tylko liczy upływ czasu wykonując w trakcie pozostałe instrukcje.

9. Omówienie przez nauczyciela czym są pamięci SRAM, EEPROM, FLASH

SRAM (ang. Static Random Access Memory), statyczna pamięć o dostępie swobodnym, jest to rodzaj pamięci półprzewodnikowej stosowanej w mikrokontrolerach i komputerach. Wykorzystywana jest jako pamięć tymczasowa do przechowywania aktualnie przetwarzanych danych.

Pamięć SRAM przechowuje dane do momentu zaniku zasilania bądź resetu układu.

EEPROM, E^{2}PROM (od ang. electrically erasable programmable read-only memory), elektrycznie kasowalna i programowalna pamięć tylko do odczytu jest to pamięć nieulotna.

Pamięć EEPROM może być kasowana i programowana tylko przy użyciu prądu elektrycznego. Liczba zapisów i kasowań jest ograniczona. Producent pamięci w mikrokontrolerze Atmega 328 znajdującym się w układzie ArduinoUNO określił żywotność na poziomie 100 000 cykli zapisu i kasowania. Po przekroczeniu tej wartości pamięć może ulec uszkodzeniu. Liczba odczytów pamięci EEPROM jest określona także na 100 000.

Pamięć flash (ang. flash memory) – rodzaj pamięci nieulotnej będącej rozwinięciem konstrukcyjnym i kontynuacją pamięci typu EEPROM. Dostęp do danych zapisanych w pamięci flash wykorzystuje stronicowanie pamięci: operacje odczytu, zapisu lub kasowania wykonywane są jednocześnie na ustalonej konstrukcyjnie liczbie komórek, pogrupowanych w strukturę będącą wielokrotnością słowa maszynowego (bajtu). Cechą wyróżniającą pamięć flash jest wykorzystanie technologii komórek wielostanowych.

W układzie Arduino UNO pamięć ta wykorzystywana jest na program, który wczytujemy do układu w ramach naszego projektu.

W układzie ArduinoUNO mikrokontroler Atmega328, w który jest on wyposażony, posiada następujące wielkości pamięci:

1. Flash 32 kB (0.5k używana jest przez tzw. bootloader)
2. SRAM 2 kB
3. EEPROM 1 kB

10. Omówienie przez nauczyciela instrukcji obsługujących pamięć EEPROM w układzie ArduinoUNO

Mikrokontroler Atmga328 posiada pamięć EEPROM o pojemności 1024 bajty.

Jej żywotność wynosi 100 000 cykli zapisu/odczytu.

Do obsługi pamięci EEPROM wykorzystamy bibliotekę EEPROM.h.

Wśród funkcji obsługujących pamięć EEPROM najważniejsze to:

- `read()` – odczytuje 1 bajt z pamięci EEPROM.
- `write()` – wprowadza do pamięci 1 bajt danych; przyjmuje dwa parametry: adres, pod który wpisujemy dane i wartość (1 bajt danych). Zapis danych trwa ok. 3,3 ms.
- `update()` – podobnie jak funkcja `write`, wprowadza dane do pamięci; ale najpierw sprawdza, czy dana, która już się znajduje pod docelowym adresem, nie jest identyczna z podaną do zapisu; zapis wykona się tylko, jeśli dane się różnią; funkcja ta jest przydatna dla ograniczenia cykli zapisów, co zwiększa żywotność pamięci EEPROM.
- `get()` – funkcja odczytująca dane z pamięci EEPROM. W odróżnieniu od funkcji `read()`, pobiera całą długość danej (obiektu).
- `put()` – funkcja wprowadza do amięci EEPROM dane wszystkich typów; posiada takie same parametry jak funkcja `write()`, jednak w odróżnieniu od niej zapisuje całą daną (obiekt) w jednym wywołaniu, a nie pojedynczy bajt jak funkcja `write()`.

11. Ćwiczenie 5 Do realizacji wraz z nauczycielem

Napisz program, w którym zmodyfikujesz projekt z ćwiczenia 4 tak, by zmiana `prog` przechowywana była w pamięci nieulotnej EEPROM.

Program powinien umożliwiać wyzerowanie wartości progu po naciśnięciu przycisku środkowego" `SW_CENTER`

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// biblioteka obsługująca pamięć nieulotną EEPROM
#include <EEPROM.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

#define SW_UP 6
#define SW_DOWN 5
#define SW_CENTER 8

// zmienna przechowująca odczyt czasu od włączenia układu
unsigned long aktCzas = 0;

// zmienna przechowuje moment włączenia diody LED
```

```
unsigned long czasLED1_on = 0;

// zmienna pomocnicza do obliczenia czasu, który upłynął od włączenia diody
unsigned long roznicaCzasu = 0;

// zmienna ”flaga” podnoszona w chwili włączenia diody LED
bool LED1_on=0;

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);
  // Ustawienie pinu diody LED jako wyjście
  pinMode(13, OUTPUT);

  // konfiguracja LCD
  lcd.begin(16, 2);
}

// utworzenie zmiennej przechowującej wartosc progu hałasu
// przypisanie do zmiennej prog wartości odczytanej z pamięci EEPROM
int prog=EEPROM.read(1);
void loop() {

  if(digitalRead(SW_DOWN) == HIGH&&prog>1)
  {
    // wpisanie do pamięci EEPROM warości progu po pomniejszeniu o 1
    // Ważne! znaki ”–” lub ”++” przed nazwą zmiennej oznaczają,
    // że operacja odejmowania/dodawania 1 wykonana będzie przed
    // wykonaniem innych działań w ramach instrukcji
      EEPROM.put(1, --prog);
  }
  if(digitalRead(SW_UP) == HIGH&&prog<1023)
  {
     // wpisanie do pamięci EEPROM warości progu po powiększeniu o 1
     EEPROM.put(1, ++prog);
  }

  // zerowanie wartości progu w chwili naciśnięcia przycisku SW_CENTER
  if(digitalRead(SW_CENTER) == HIGH)
  {
    int zero=0;
    EEPROM.put(1, zero);
    prog=0;
  }

  // poniżej wyświetlam wartość progu
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Prog halasu:");
```

```
  lcd.setCursor(0, 1);
  // wyświetlenie wartości progu hałasu odczytanej z pamięci EEPROM
  lcd.print(EEPROM.get(1, prog));

  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.println("Poziom halasu");
  display.setCursor(0,7);
  display.println("wynosi");
  display.setCursor(0,14);
  // wyświetlenie wartości odczytanej z mikrofonu.
  display.println(analogRead(A0));

  display.setCursor(0,21);
  display.println(millis());
  display.display();
  // instrukcja warunkowa sprawdzająca,
  // czy hałasu nie jest większa niż wartość progu
  // wtedy podnoszona jest ”flaga” włączenia diody LED
  // i przypisywany jest czas w którym to włączenie nastąpiło

  if(prog<=analogRead(A0))
  {
    LED1_on=1;
    czasLED1_on = millis();
  }

  // jeśli ”flaga” włączenia diody jest podniesiona
  // i różnica czasu, który minał od włączenia jest mniejsza niż 1 sekunda
  // to włącz diodę LED D1 i zmierz czas
  // w przeciwnym wypadku wyłącz diodę LED, opuść ”flagę” i wyzeruj licznik czasu
  if(LED1_on==1 && roznicaCzasu<1000)
  {
    // Pobierz czas, który minął od startu programu
    aktCzas = millis();
    roznicaCzasu = aktCzas - czasLED1_on;
    digitalWrite(13, HIGH);
  }
  else
  {
    digitalWrite(13, LOW);
    LED1_on=0;
    roznicaCzasu=0;
  }
}
```

W programie w chwili, kiedy następują zmiany wartości zmiennej `prog`, od razu zostają przekazane do pamięci EEPROM.

Takie działanie pozwala nam na odtworzenie wartości z pamięci nieulotnej po zaniku zasilania.

Działanie to jest szczególnie pomocne w sytuacjach, kiedy mamy jakieś dane konfiguracyjne układu, które są możliwe do zmiany przez użytkownika układu i powinny być zachowane w kolejnych uruchomieniach układu po zaniku zasilania czy restarcie systemu. Takimi danymi konfiguracyjnymi, mogą być np.:

- dane o uprawnionych użytkownikach układu
  - telefon
  - login
  - PIN/hasło
- prędkość (częstotliwość) transmisji
- informacje o włączeniu/wyłączeniu poszczególnych komponentów systemu „inteligentnego domu”
- etc.

12. Przeprowadzenie quizu podsumowującego lekcję.

1. Czujnik dźwięku został dołączony w zestawie edukacyjnym TME-EDU-ARD-2 do układu ArduinoUNO do wejścia:
   a) A0
   b) A1
   c) A2
   d) A3
2. Odczytana wartość wejścia analogowego Arduino UNO z podłączonym czujnikiem dźwięku jest:
   a) podana w skali od 0 do nieskończoności
   b) podana w skali od 0 do 100
   c) podana w skali od 0 do 1023
   d) podana w skali od –100 do 100
3. Zmienne w programach na ArduinoUNO zapisywane są w pamięci:
   a) HDD
   b) FLASH
   c) SRAM
   d) EEPROM
4. Tzw. ulotną pamięcią jest:
   a) HDD
   b) FLASH
   c) SRAM
   d) EEPROM
5. By zachować dane po zaniku zasilania w układzie ArduinoUNO bez dołączania dodatkowych układów, możemy wykorzystać:
   a) HDD
   b) SSD

c) SRAM
d) EEPROM

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który odczyta wartość z czujnika dźwięku i wyświetli poziom głośności na wyświetlaczu OLED.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED

// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

void setup() {
 display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
 delay(500);
}

void loop() {
   display.clearDisplay();
   display.setTextColor(WHITE);
   display.setCursor(0,0);
   display.println("Poziom halasu");
   display.setCursor(0,7);
   display.println("wynosi");
   display.setCursor(0,14);
   // wyświetlenie wartości odczytanej z mikrofonu.
   display.println(analogRead(A0));
   display.display();
   delay(200);
}
```

Ćwiczenie 2

Napisz program, który odczyta wartość z czujnika dźwięku i jeśli poziom hałasu przekroczy wartość 500, to włączona zostanie dioda led D1 na 5 sekund.

Oceń, czy odczytany z układu poziom hałasu o wartości 500 może być już uznany za głośny i dokuczliwy.

Ćwiczenie 3

Napisz program, w którym zmodyfikujesz projekt z ćwiczenia 2 tak, by próg załączania diody LED D1 był konfigurowany przez przyciski monostabilne „góra”/„dół”. Wartość progu załączania diody D1 powinna być wyświetlana na wyświetlaczu LCD, a poziom hałasu na wyświetlaczu OLED.

Skróć czas świecenia diody LED D1 do 1 sekundy po wykryciu przekroczenia progu.

Odpowiedz w grupie na poniższe pytania:

- Jakie zastosowanie może mieć czujnik poziomu dźwięku w centrali sterowania inteligentnym domem?
- Jakie niedoskonałości w pomiarach natężenia dźwięku widzimy?
- Czy mierzenie chwilowej wartości dźwięku jest dobrym rozwiązaniem?
- Jakie możesz zaproponować inne możliwości pomiaru i analizy poziomu dźwięku (hałasu)? Np. średnia z 10 pomiarów wykonanych co 10 ms. Podaj wady i zalety tych rozwiązań.

Czy mierzenie chwilowej wartości dźwięku może być przydatne?

Ćwiczenie 4

Napisz program, w którym zmodyfikujesz projekt z ćwiczenia 3 tak, by zmiana progu nie była opóźniona funkcją `delay()`.

Ćwiczenie 5

Napisz program, w którym zmodyfikujesz projekt z ćwiczenia 4 tak, by zmiana `prog` przechowywana była w pamięci nieulotnej EEPROM.

Program powinien umożliwiać wyzerowanie wartości progu po naciśnięciu przycisku środkowego” `SW_CENTER`

Quiz

1. Czujnik dźwięku został dołączony w zestawie edukacyjnym TME-EDU-ARD-2 do układu ArduinoUNO do wejścia:
   a) A0
   b) A1
   c) A2
   d) A3
2. Odczytana wartość wejścia analogowego Arduino UNO z podłączonym czujnikiem dźwięku jest:
   a) podana w skali od 0 do nieskończoności
   b) podana w skali od 0 do 100
   c) podana w skali od 0 do 1023
   d) podana w skali od –100 do 100
3. Zmienne w programach na ArduinoUNO zapisywane są w pamięci:
   a) HDD
   b) FLASH
   c) SRAM
   d) EEPROM
4. Tzw. ulotną pamięcią jest:
   a) HDD
   b) FLASH
   c) SRAM
   d) EEPROM
5. By zachować dane po zaniku zasilania w układzie ArduinoUNO bez dołączania dodatkowych układów, możemy wykorzystać:
   a) HDD
   b) SSD
   c) SRAM
   d) EEPROM

Lekcja 14

Sterujemy zestawem edukacyjnym TME-EDU-ARD-2 za pomocą pilota IR.

Czas trwania lekcji: **45 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych oraz pętli w języku C++.
Utrwalenie umiejętności odczytu danych z potencjometru, przycisków monostabilnych, czujnika temperatury i czujnika natężenia światła w zestawie edukacyjnym TME-EDU-ARD-2.
Utrwalenie umiejętności pisania programów działających z wyświetlaczami LCD, graficznym OLED i LED.
Utrwalenie umiejętności tworzenia programów z użyciem wielozadaniowości w pracy mikrokontrolera.

Efekty kształcenia

- Uczeń potrafi stosować zmienne w języku C++.
- Uczeń stosuje w swoich programach potencjometr oraz przyciski monostabilne do wprowadzania danych.
- Uczeń stosuje w swoich programach wyświetlacze LCD, graficzny OLED i LED.
- Uczeń w swoich programach używa danych odczytanych z czujnika temperatury.
- Uczeń w swoich programach używa danych odczytanych z czujnika natężenia światła.
- Uczeń stosuje wielozadaniowość w swoich programach.
- Uczeń stosuje w swoich programach komunikację z pilotem IR w standardzie RC5.

Wstęp

Podczas dotychczasowych lekcji poznaliśmy wiele sposobów nawiązania interakcji w naszych projektach z użytkownikiem. Z układem TME-EDU-ARD-2 użytkownik może komunikować się wykorzystując transmisję szeregową UART, przyciski monostabilne, potencjometr, czy nawet przez odczyty danych z podłączonych czujników (np. czujnik dźwięku- wykonanie polecenia po klaśnięciu) . Jednak to nie są wszystkie metody jakimi użytkownik może wprowadzać dane do układu i nawiązywać z nim komunikację.

Na tej lekcji poznamy jeszcze jeden sposób interakcji, za pomocą pilota podczerwieni (IR). Zestaw edukacyjny wyposażony jest w scalony odbiornik podczerwieni TSOP2236, który może odbierać dane z pilota działającego w standardzie RC5.

Wykorzystanie pilota da nam po pierwsze możliwość wprowadzenia do układu sterowania znacznie większą ilością przycisków niż te dostępne w samym zestawie edukacyjnym (np. przeciętny pilot TV ma 20–30 przycisków, które możemy zaprogramować). Po drugie, sterowanie podczerwienią to możliwość sterowania układem z pewnej odległości. Nie musimy mieć bezpośredniego kontaktu z płytką edukacyjną. Wystarczy, że będziemy ją mieli w zasięgu wzroku.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

–

- Jak programować poznane na poprzednich lekcjach podzespoły umożliwiające wprowadzanie danych, tj. Przyciski monostabilne, potencjometr, czujnik temperatury, czujnik światła, czujnik dźwięku.
- Powtórzenie wiadomości o sposobie sterowania i wyświetlania danych na wyświetlaczu LCD, LED i wyświetlaczu graficznym OLED.
- Powtórzenie wiadomości o możliwości zaprogramowania dźwięków z buzzera
- Powtórzenie wiadomości na temat zapisywanie danych w pamięci nieulotnej EEPROM.

3. Omówienie przez nauczyciela sposobu działania odbiornika podczerwieni.

W naszym zestawie został wbudowany scalony odbiornik podczerwieni TSOP2236, który może odbierać sygnały z pilotów IR działających w standardzie RC5. Standard RC5 nie jest nowością na rynku, a piloty działające właśnie w tym standardzie są wśród nas wszechobecne. Telewizory, tunery, dekodery, urządzenia AV – wszystkie one poza przyciskami obecnymi na obudowie i na panelu sterowania mają możliwość sterowania pilotem. Wśród najnowszych urządzeń możliwe jest, że na wyposażeniu dostaliśmy piloty tzw. radiowe. Ich nasz zestaw edukacyjny nie obsłuży. Jednak wszystkie pozostałe i ogromna większość starszych urządzeń u nas w domu używa właśnie pilota podczerwieni działającego w standardzie RC5.

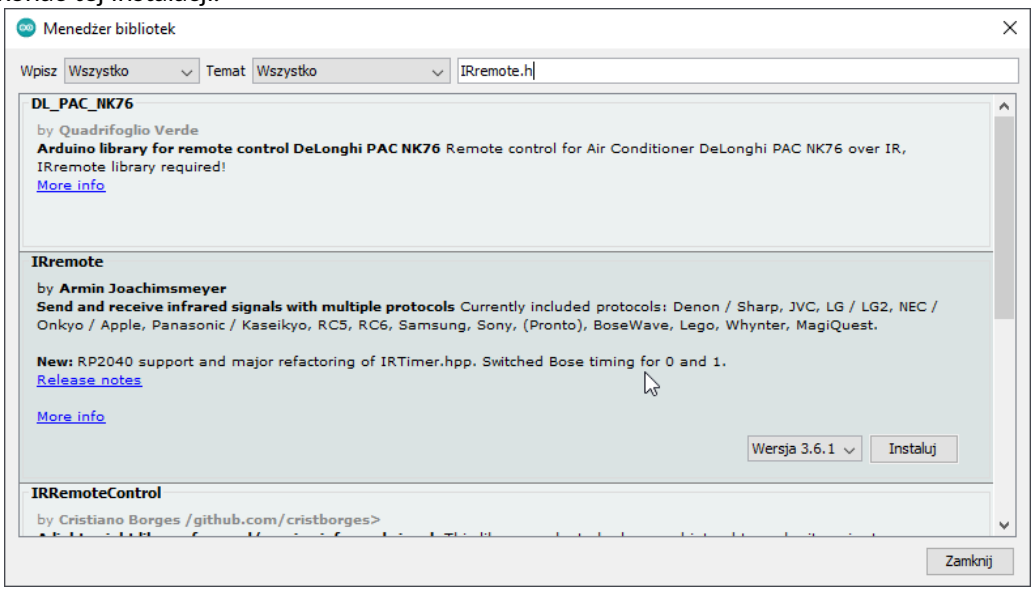
Chcąc więc zaprogramować nasz układ na interakcję z takim pilotem nie musimy więc wydawać dodatkowych pieniędzy, a możemy wykorzystać to co już mamy dołączone do naszych domowych odbiorników.

W naszych ćwiczeniach będziemy mieli możliwość zaprogramowania dowolnego pilota. A przykłady, które będziemy wykonywać, będą realizowane w oparciu o pilot do telewizora, czyli pilot z możliwością wybrania m.in. konkretnego kanału (cyfry 0–9) przycisków +kanał/–kanał, +głośność/–głośność, zasilanie.

Pilot działający w obsługiwanym standardzie RC5 przesyła ramkę danych o rozmiarze 14 bitów. Dane te mogą być odebrane przez nasz program i przetworzone na kod liczbowy. Kod liczbowy, jaki otrzymamy po odebraniu sygnału z pilota, najczęściej będziemy prezentowali w systemie szesnastkowym (HEX – heksadecymalnym).

Do obsługi pilota IR będziemy potrzebowali nowej biblioteki, która może nie być od razu zainstalowana w naszym środowisku ArduinoIDE. Sprawdźmy więc w naszym Menedżerze bibliotek, czy nie musimy jej pobrać. Po wejściu do Menedżera bibliotek wpisujemy w pole wyszukiwania „IRremote.h". Powinniśmy jako jedną z pierwszych pozycji zobaczyć właśnie paczkę

bibliotek IRremote. Jeśli tak jak na poniższym zrzucie ekranu wyświetla nam się przycisk „Instaluj", to musimy dokonać tej instalacji.



Po dołączeniu do naszego programu biblioteki `#include <IRremote.h>` powinniśmy utworzyć obiekt naszego pilota i zmienną, w której przechowamy dane z niego pochodzące. Zrobimy to za pomocą dwóch poleceń umieszczonych bezpośrednio pod deklaracjami bibliotek: `IRrecv irrecv(3); decode_results results;`

W części konfiguracyjnej programu `void setup()` powinniśmy uruchomić nasz odbiornik podczerwieni poleceniem `irrecv.enableIRIn();`

Od tej chwili możemy sprawdzać, czy w buforze danych odbiornika IR znajdują się jakieś dane – `irrecv.decode(&results)`.

Odebrane dane znajdują się w zmiennej `results`, jednak żeby je poprawnie odczytać, musimy użyć polecenia `results.value`.

Po odebraniu danych z czujnika musimy wznowić jego działanie poleceniem `irrecv.resume();`

4. Ćwiczenie 1 Do realizacji wraz z nauczycielem.

Napisz program, który odczyta kody przycisków odbieranych z twojego pilota podczerwieni i wyświetli je na wyświetlaczu OLED. Kody powinny być przedstawione w postaci liczby szesnastkowej.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED
// biblioteka obsługująca czujnik podczerwieni
#include <IRremote.h>

// tworzymy obiekt/zmienną pilota IR
IRrecv irrecv(3);

// tworzymy obiekt/zmienną do przechowywania danych odczytywanych z pilota IR
decode_results results;

// utworzenie obiektu wyświetlacza OLED
```

```
Adafruit_SSD1306 display(NULL);

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);

  // włączamy odbiornik podczerwieni
  irrecv.enableIRIn();
}

void loop() {
  display.clearDisplay();
  display.setTextColor(WHITE);

  // instrukcja warunkowa sprawdza czy odebrane zostały dane z czujnika podczerwieni
  // jeśli tak to dane zostaną zapisane w obiekcie results
  if (irrecv.decode(&results)) {
    display.setCursor(0,0);
    // przekazujemy dane z obiektu results(czujnika podczerwienie) na wyświetlacz OLED
    // dane są formatowane do liczby HEX, czyli wartości szczesnastkowej
    display.println(results.value, HEX);
    // wznawiamy odbiór danych z czujnika podczerwieni
    irrecv.resume(); // wznów odbieranie
  }
  display.display();
  delay(100);
}
```

W programie w części nagłówkowej dołączono bibliotekę `IRremote.h` i utworzono obiekty, dzięki którym możemy kontrolować odbiornik podczerwieni `IRrecv irrecv(3);`. Dodatkowo konieczne było utworzenie obiektu, w którym będziemy przechowywać dane od odbiornika podczerwieni `decode_results results;`.

W części konfiguracyjnej funkcji `void setup()` użyliśmy instrukcji `irrecv.enableIRIn();`, która spowodowała włączenie czuwania odbiornika podczerwieni.

W funkcji `void loop()` sprawdzamy, czy do odbiornika dotarł jakiś sygnał. Jeśli tak, to funkcja `irrecv.decode(&results)` zwróci prawdę i umieści odebrany kod pod adresem obiektu `results`. Za pomocą instrukcji `results.value` możemy pobrać kod w formie liczby całkowitej. W naszym przypadku instrukcja ta zwraca liczbę całkowitą i w formacie liczby szesnastkowej przekazuje do wyświetlenia na wyświetlaczu OLED. Po operacji pobrania danych z odbiornika konieczne jest wznowienie pracy odbiornika poleceniem `irrecv.resume();`.

5. Ćwiczenie 2 Do realizacji samodzielnej przez uczniów.

Wyświetlając kody przycisków z poprzedniego ćwiczenia spisz wartości kodów dla przycisków +głośność/−głośność i zasilanie. Następnie napisz program, który odbierze sygnały z pilota i jeśli przyciśnięty jest przycisk +głośność/−głośność, zmieni wartość cyfry wyświetlanej na wyświetlaczu LCD.

Przykładowy kod programu, gdzie zastosowano pilot do telewizora Sharp GA586WJSA:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>
// biblioteka obsługująca czujnik podczerwieni
#include <IRremote.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// tworzymy obiekt/zmienną pilota IR
IRrecv irrecv(3);

// tworzymy obiekt/zmienną do przechowywania danych odczytywanych z pilota IR
decode_results results;

void setup() {
  // konfiguracja LCD
  lcd.begin(16, 2);

  // włączamy odbiornik podczerwieni
  irrecv.enableIRIn();
  delay(200);
}
// tworzymy zmienną typu całkowitego do przechwoawnia kodu przycisku pilota IR
long odczyt;

// zmienna, która będzie wyświetlana na wyświetlaczy LCD
long licznik=0;
void loop() {
  // instrukcja warunkowa sprawdza czy odebrane zostały dane z czujnika podczerwieni
  // jeśli tak to dane zostaną zapisane w obiekcie results
  if (irrecv.decode(&results)) {
    // zapisujemy odczytaną wartość kodu z pilota IR do zmiennej odczyt
    odczyt=results.value;
    // wznawiamy odbiór danych z czujnika podczerwieni
    irrecv.resume(); // wznów odbieranie
  }
  else
  {
    // jeśli naciśnięto inny przycisk
    // bądź w ogóle nie naciśnięto przycisku zerujemy zmienną odczyt
    odczyt=0;
  }

  // liczba 0x7E16B93A to zapis szesnastkowy kodu przycisku +głośność w testowanym rzykładowym pilocie
  // jeśli odczytany kod z odbiornika podczerwieni to właśnie kod tego przycisku to dodajemy do licznika 1
  if(odczyt==0x7E16B93A)
  {
    licznik++;
```

```
  }
  // Liczba 0x1BE8C80d to zapis szesnastkowy kodu przycisku -głośność
  // w testowanym przykładowym pilocie.
  // Jeśli odczytano z odbiornika podczerwieni ten właśnie kod, to zmniejszamy licznik o 1
  if(odczyt==0x1BE8C80d)
  {
    licznik--;
  }
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Odczyt:");
  lcd.setCursor(0, 1);
  // wyświetlamy aktualną wartość licznika
  lcd.print(licznik);
  delay(100);
}
```

W programie, podobnie jak w ćwiczeniu 1, wprowadziliśmy bibliotekę obsługującą odbiornik podczerwieni, utworzyliśmy potrzebne obiekty i zmienne. Przyjmowane kody przekazujemy nie bezpośrednio na wyświetlacz, tylko do zmiennej całkowitej `odczyt` typu `long`.

Następnie w dwóch instrukcjach warunkowych sprawdzamy, czy wartości w zmiennej `odczyt` pokrywają się z wartościami spisanymi dla przycisków +głośność/–głośność. Jeśli tak, to odpowiednio zwiększamy lub zmniejszamy licznik i przekazujemy uzyskaną wartość do wyświetlenia na wyświetlaczu LCD.

6. Ćwiczenie 3 Do realizacji samodzielnej przez uczniów.

Napisz program, który wyświetli na wyświetlaczu LED cyfrę wskazaną przez twój pilot TV.

Przykładowy kod programu, gdzie zastosowano pilot do telewizora Sharp GA586WJSA:

```
// biblioteka obsługująca czujnik podczerwieni
#include <IRremote.h>
#include "Adafruit_MCP23008.h"// biblioteka LED

// utworzenie obiektu wyświetlacza LED
Adafruit_MCP23008 seg7;

// utworzenie tablicy z cyframi dla wyświetlacza LED
uint8_t digit[10] = {
 B00111111, // ”0”
 B00000110, // ”1”
 B01011011, // ”2”
 B01001111, // ”3”
 B01100110, // ”4”
 B01101101, // ”5”
 B01111101, // ”6”
 B00000111, // ”7”
```

```
 B01111111, // "8"
 B01101111, // "9"
};
// w tablicy kody[10] będziemy pamiętać 10 kodów przycisków
// reprezentujących cyfry na testowanym pilocie telewizyjnym IR
long kody[10];

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// tworzymy obiekt/zmienną pilota IR
IRrecv irrecv(3);

// tworzymy obiekt/zmienną do przechowywania danych odczytywanych z pilota IR
decode_results results;

void setup() {
  // do kolejnych pozycji w tablicy kody przypisujemy spisane wcześniej kody cyfr;
  kody[0]=0xAB9B313A;
  kody[1]=0xA1F51F22;
  kody[2]=0xF3A9C948;
  kody[3]=0xA34F5A01;
  kody[4]=0x22614D75;
  kody[5]=0x96C5A8AB;
  kody[6]=0x44ABD1FD;
  kody[7]=0x1A5C4D;
  kody[8]=0x57A67691;
  kody[9]=0x9543D7CF;
   // konfiguracja pinów ekspandera wyświetlacza LED
   seg7.begin(0x4);
   seg7.pinMode(0, OUTPUT);
   seg7.pinMode(1, OUTPUT);
   seg7.pinMode(2, OUTPUT);
   seg7.pinMode(3, OUTPUT);
   seg7.pinMode(4, OUTPUT);
   seg7.pinMode(5, OUTPUT);
   seg7.pinMode(6, OUTPUT);
   seg7.pinMode(7, OUTPUT);

  // włączamy odbiornik podczerwieni
  irrecv.enableIRIn();
  delay(200);
}
// tworzymy zmienną typu całkowitego do przechwoawnia kodu przycisku pilota IR
long odczyt;

// zmienna, która będzie wyświetlana na wyświetlaczy LCD
long liczba=0;
```

```
void loop() {
  // instrukcja warunkowa sprawdza, czy odebrane zostały dane z czujnika podczerwieni
  // jeśli tak, to dane zostaną zapisane w obiekcie results
  if (irrecv.decode(&results)) {
    // zapisujemy odczytaną wartość kodu z pilota IR do zmiennej odczyt
    odczyt=results.value;
    // wznawiamy odbiór danych z czujnika podczerwieni
    irrecv.resume(); // wznów odbieranie
  }

  // w poniższej pętli przeszukiwana jest tablica z zapisanymi kodami przycisków cyfr
  for(int i=0; i<10;i++)
  {
    if(odczyt==kody[i])
    {
      liczba=i;
    }
  }
  // na wyświetlaczy LED wyświetlana jest liczba zgodna ze wskazaniem pilota IR
  seg7.writeGPIO(digit[liczba]);
  delay(100);
}
```

Do identyfikacji kodów kolejnych przycisków wykorzystujemy w programie tablicę liczb całkowitych `kody[10]`. W części konfiguracyjnej funkcji `void setup()` przypisujemy do tej tablicy wartości kodów przycisków pilota od 0 do 9, spisanych w poprzednich ćwiczeniach.

Po odebraniu kodu z odbiornika podczerwieni w pętli `for(int i=0; i<10;i++)`, sprawdzamy, czy odczytany kod jest taki sam jak jeden z kodów zapisanych w tablicy `kody[]`. Jeśli tak, to wartość indeksu pętli stanowi w tym przypadku wartość liczby, którą wyświetlamy na wyświetlaczu LED.

7. Ćwiczenie 4 Do realizacji samodzielnej przez uczniów.

Napisz program, który do programu z ćwiczenia 3 doda tryb „nauki/konfiguracji” przycisków pilota. Tryb ten będzie się włączał, jeśli przytrzymany zostanie przez 5 sekund przycisk monostabilny środkowy `SW_CENTER`.

Instrukcje konfiguracji powinny wyświetlać się na wyświetlaczu LCD. Przykładowe komunikaty i polecenia wyświetlane na wyświetlaczu LED:

- „Witamy w trybie konfiguracji”
- „Wcisnij wartosc 0 na pilocie”
- „Wcisnij wartosc 1 na pilocie”
- „Konfiguracja zakończona”

Przykładowy kod programu, gdzie zastosowano pilot do telewizora Sharp GA586WJSA:

```
#include <Wire.h>
#include <hd44780.h>
```

```
#include <hd44780ioClass/hd44780_I2Cexp.h>
// biblioteka obsługująca czujnik podczerwieni
#include <IRremote.h>
#include "Adafruit_MCP23008.h"// biblioteka LED

// utworzenie obiektu wyświetlacza LED
Adafruit_MCP23008 seg7;

// utworzenie tablicy z cyframi dla wyświetlacza LED
uint8_t digit[10] = {
 B00111111, // ”0”
 B00000110, // ”1”
 B01011011, // ”2”
 B01001111, // ”3”
 B01100110, // ”4”
 B01101101, // ”5”
 B01111101, // ”6”
 B00000111, // ”7”
 B01111111, // ”8”
 B01101111, // ”9”
};

// w tablicy kody[10] będziemy pamiętać 10 kodów przycisków
// reprezentujących cyfry na testowanym pilocie telewizyjnym IR
long kody[10];

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// tworzymy obiekt/zmienną pilota IR
IRrecv irrecv(3);

// tworzymy obiekt/zmienną do przechowywania danych odczytywanych z pilota IR
decode_results results;

// zmienna przechowująca odczyt czasu od włączenia układu
unsigned long aktCzas = 0;

// zmienna przechowuje moment włączenia przycisku SW_CENTER
unsigned long czasCENTER_on = 0;

// zmienna pomocnicza do obliczenia czasu, który upłynął od włączenia diody
unsigned long roznicaCzasu = 0;

bool SW_CENTER_on=0;

#define SW_CENTER 8

void setup() {
   // konfiguracja pinów ekspandera wyświetlacza LED
   seg7.begin(0x4);
```

```
    seg7.pinMode(0, OUTPUT);
    seg7.pinMode(1, OUTPUT);
    seg7.pinMode(2, OUTPUT);
    seg7.pinMode(3, OUTPUT);
    seg7.pinMode(4, OUTPUT);
    seg7.pinMode(5, OUTPUT);
    seg7.pinMode(6, OUTPUT);
    seg7.pinMode(7, OUTPUT);

  // włączamy odbiornik podczerwieni
  irrecv.enableIRIn();
  // konfiguracja LCD
  lcd.begin(16, 2);
  delay(200);
}
// tworzymy zmienną typu całkowitego do przechwoawnia kodu przycisku pilota IR
long odczyt;

// zmienna, która będzie wyświetlana na wyświetlaczu LCD
long liczba=0;
void loop() {
  // instrukcja warunkowa sprawdza czy odebrane zostały dane z czujnika podczerwieni
  // jeśli tak to dane zostaną zapisane w obiekcie results
  if (irrecv.decode(&results)) {
    // zapisujemy odczytaną wartość kodu z pilota IR do zmiennej odczyt
    odczyt=results.value;
    // wznawiamy odbiór danych z czujnika podczerwieni
    irrecv.resume(); // wznów odbieranie
  }

  // w poniższej pętli przeszukiwana jest tablica z zapisanymi kodami przycisków cyfr
  for(int i=0; i<10;i++)
  {
    if(odczyt==kody[i])
    {
      liczba=i;
    }
  }
  // na wyświetlaczy LED wyświetlana jest liczba zgodna ze wskazaniem pilota IR
  seg7.writeGPIO(digit[liczba]);

  // jeżeli przycisk SW_CENTER jest wciśnięty to włączam licznik czau
  if(digitalRead(SW_CENTER) == HIGH)
  {
    if(SW_CENTER_on==0)
    {
      czasCENTER_on = millis();
```

```
      SW_CENTER_on=1;
    }
    else
    {
      aktCzas = millis();
      roznicaCzasu = aktCzas - czasCENTER_on;
    }
  }
  else
  {
    SW_CENTER_on=0;
  }
  // jeżeli od przyciśnięcia przycisku SW_CENTER minęło 5 sekund,
  // to wykonujemy sekwencję wyświetleń komunikatów na wyświetlaczu LCD
  // i pobieramy kody kolejnych cyfr z pilota IR
  if(roznicaCzasu>5000)
  {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Witamy w trybie ");
    lcd.setCursor(0, 1);
    lcd.print("konfiguracji");
    delay(1000);
    for(int i=0; i<10;i++)
    {
      kody[i]=0;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Wcisnij wartosc");
      lcd.setCursor(0, 1);
      lcd.print(i);
      lcd.setCursor(3, 1);
      lcd.print("na pilocie");
      while (kody[i]==0)
      {
        if(irrecv.decode(&results)) {
          // zapisujemy odczytaną wartość kodu z pilota IR do zmiennej odczyt
          kody[i]=results.value;
          // wznawiamy odbiór danych z czujnika podczerwieni
          irrecv.resume(); // wznów odbieranie
        }
        }
      }
      delay(200);
      while (irrecv.decode(&results)) {
```

```
            irrecv.resume(); // wznów odbieranie
            }
          }
          delay(1000);
        }
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Konfiguracja ");
        lcd.setCursor(0, 1);
        lcd.print("zakończona");
        delay(1000);
        SW_CENTER_on=0;
        roznicaCzasu=0;
      }
      else
      {
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("Konfiguruj");
        lcd.setCursor(0, 1);
        lcd.print("SW_CENTER");
      }
      delay(100);
    }
```

W przyjętym rozwiązaniu, aby sprawnie i bez przerw następowały odczyty z przycisków monostabilnych i odczyty z czujnika podczerwieni, zastosowaliśmy wielozadaniowość. Dla tego celu wprowadziliśmy zmienne `unsigned long aktCzas`, `unsigned long czasCENTER_on`, `unsigned long roznicaCzasu`, które kontrolują czas nieprzerwanego naciśnięcia przycisku `SW_CENTER`. Jeśli różnica czasu od włączenia i nieprzerwanego trzymania przycisku była większa od 5000 (ms), to rozpoczynamy sekwencję odczytu danych z odbiornika podczerwieni.

W tym celu zastosowano pętlę `for(int i=0; i<10;i++)` która ma dokonywać odczytu z odbiornika podczerwieni – aż do momentu otrzymania kodu przycisku. Wtedy też ppobrany kod przypisujemy do tablicy z kodami `kody[i]=results.value;` i wznawiamy odczyt dla nowego przycisku (nowej cyfry).

Po odebraniu danych o kodach wszystkich przycisków program przechodzi do standardowego trybu pracy, czyli do sprawdzania, czy odbierane kody na odbiorniku podczerwieni są zgodne z tymi zapisanymi w tablicy `kody[]`. Jeśli tak, to wyświetlamy odpowiednią cyfrę (zgodną z indeksem w tablicy `kody[]`) na wyświetlaczu LED.

8. Przeprowadzenie quizu podsumowującego lekcję.

1. Do pracy z odbiornikiem podczerwieni niezbędne będzie użycie biblioteki:
   a) Adafruit_MCP23008.h
   b) IRremote.h
   c) Wire.h

d) hd44780.h

2. Odbiornik podczerwieni TSOP2236 współpracuje z pilotami podczerwieni działającymi w standardzie:
   a) RC1
   b) RC3
   c) RC5
   d) RC7
3. Tworząc programy współpracujące z odbiornikiem podczerwieni możemy odebrać sygnał
   a) Z max 5 przycisków pilota
   b) Z max 20 przycisków pilota
   c) Z max 50 przycisków pilota
   d) Ponad 50 przycisków pilota
4. Funkcja, która sprawdza przyjęcie danych i odczytuje je z odbiornika podczerwieni to:
   a) `irrecv.decode()`
   b) `results.value()`
   c) decode_results()
   d) `IRrecv irrecv()`
5. Aby włączyć odbiornik podczerwieni użyjemy polecenia:
   a) `irrecv.decode(&results);`
   b) `irrecv.resume();`
   c) `irrecv.enableIRIn();`
   d) `results.value;`

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który odczyta kody przycisków odbieranych z twojego pilota podczerwieni i wyświetli je na wyświetlaczu OLED. Kody powinny być przedstawione w postaci liczby szesnastkowej.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Adafruit_SSD1306.h>// blbliteka OLED
// biblioteka obsługująca czujnik podczerwieni
#include <IRremote.h>

// tworzymy obiekt/zmienną pilota IR
IRrecv irrecv(3);

// tworzymy obiekt/zmienną do przechowywania danych odczytywanych z pilota IR
decode_results results;

// utworzenie obiektu wyświetlacza OLED
Adafruit_SSD1306 display(NULL);

void setup() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C, false);
  delay(500);

  // włączamy odbiornik podczerwieni
  irrecv.enableIRIn();
}

void loop() {
  display.clearDisplay();
  display.setTextColor(WHITE);

  // instrukcja warunkowa sprawdza czy odebrane zostały dane z czujnika podczerwieni
  // jeśli tak to dane zostaną zapisane w obiekcie results
  if (irrecv.decode(&results)) {
    display.setCursor(0,0);
    // przekazujemy dane z obiektu results(czujnika podczerwienie) na wyświetlacz OLED
    // dane są formatowane do liczby HEX, czyli wartości szczesnastkowej
    display.println(results.value, HEX);
    // wznawiamy odbiór danych z czujnika podczerwieni
    irrecv.resume(); // wznów odbieranie
  }
```

```
    display.display();
    delay(100);
}
```

Ćwiczenie 2

Wyświetlając kody przycisków z poprzedniego ćwiczenia spisz wartości kodów dla przycisków +głośność/–głośność i zasilanie. Następnie napisz program, który odbierze sygnały z pilota i jeśli przyciśnięty jest przycisk +głośność/–głośność, zmieni wartość cyfry wyświetlanej na wyświetlaczu LCD.

Ćwiczenie 3

Napisz program, który wyświetli na wyświetlaczu LED cyfrę wskazaną przez twój pilot TV.

Ćwiczenie 4

Napisz program, który do programu z ćwiczenia 3 doda tryb „nauki/konfiguracji" przycisków pilota. Tryb ten będzie się włączał, jeśli przytrzymany zostanie przez 5 sekund przycisk monostabilny środkowy `SW_CENTER`.

Instrukcje konfiguracji powinny wyświetlać się na wyświetlaczu LCD. Przykładowe komunikaty i polecenia wyświetlane na wyświetlaczu LED:

- „Witamy w trybie konfiguracji"
- „Wcisnij wartosc 0 na pilocie"
- „Wcisnij wartosc 1 na pilocie"
- „Konfiguracja zakończona"

Quiz

1. Do pracy z odbiornikiem podczerwieni niezbędne będzie użycie biblioteki:
   a) Adafruit_MCP23008.h
   b) IRremote.h
   c) Wire.h
   d) hd44780.h
2. Odbiornik podczerwieni TSOP2236 współpracuje z pilotami podczerwieni działającymi w standardzie:
   a) RC1
   b) RC3
   c) RC5
   d) RC7
3. Tworząc programy współpracujące z odbiornikiem podczerwieni możemy odebrać sygnał
   a) Z max 5 przycisków pilota
   b) Z max 20 przycisków pilota
   c) Z max 50 przycisków pilota
   d) Ponad 50 przycisków pilota
4. Funkcja, która sprawdza przyjęcie danych i odczytuje je z odbiornika podczerwieni to:
   a) `irrecv.decode()`
   b) `results.value()`
   c) decode_results()
   d) `IRrecv irrecv()`
5. Aby włączyć odbiornik podczerwieni użyjemy polecenia:
   a) `irrecv.decode(&results);`
   b) `irrecv.resume();`
   c) `irrecv.enableIRIn();`
   d) `results.value;`

Lekcja 15

Elementy programowania obiektowego w oparciu o zestaw edukacyjny TME-EDU-ARD-2. Tworzymy prostą grę.

Czas trwania lekcji: **90 min.**

Cele kształcenia

Utrwalenie wiedzy nt. stosowania zmiennych i instrukcji warunkowych oraz pętli w języku C++.
Utrwalenie umiejętności odczytu danych z potencjometru, przycisków monostabilnych, transmisji UART w zestawie edukacyjnym TME-EDU-ARD-2.
Utrwalenie umiejętności pisania programów działających z wyświetlaczami LCD i graficznym OLED.
Zapoznanie z elementami programowania obiektowego.

Efekty kształcenia

- Uczeń potrafi stosować zmienne w języku C++.
- Uczeń stosuje w swoich programach potencjometr oraz przyciski monostabilne do wprowadzania danych.
- Uczeń stosuje w swoich programach wyświetlacze LCD, graficzny OLED.
- Uczeń komunikuje się z układem ArduinoUNO wykorzystując transmisję UART
- Uczeń stosuje w swoich programach programowanie obiektowe.

Wstęp

Proponowana lekcja wprowadzi Cię do zagadnień programowania obiektowego. W trakcie lekcji dowiesz się, czym jest klasa, pola, metody, konstruktor, dane i metody prywatne, dane i metody publiczne. Wszystkie te zagadnienia będą oparte na ćwiczeniach, które są do wykonania na zestawie edukacyjnym TME-EDU-ARD-2.

W tej lekcji opierać się będziemy na podzespołach zestawu edukacyjnego i zasobach Arduino UNO, które poznaliśmy już wcześniej, takich jak: przyciski monostabilne, potencjometr czy transmisja UART. Dane i efekty naszych prac wyświetlimy na wyświetlaczu LCD i OLED.

Ćwiczenia, w których będziemy poznawać elementy programowania obiektowego, doprowadzą nas do ćwiczenia, w którym stworzymy prostą grę w „kółko i krzyżyk”, gdzie na wyświetlonej planszy rozgrywkę będzie prowadzić użytkownik i układ ArduinoUNO.

Przebieg lekcji

1. Przedstawienie celów lekcji.

2. Powtórzenie wiadomości z poprzednich lekcji,

–

- Jak programować poznane na poprzednich lekcjach podzespoły umożliwiające wprowadzanie danych, tj. przyciski monostabilne, potencjometr, czujnik temperatury, czujnik światła.
- Powtórzenie wiadomości o sposobie sterowania i wyświetlania danych na wyświetlaczu LCD i wyświetlaczu graficznym OLED.
- Powtórzenie wiadomości o tym, jak w programach na Arduino UNO możemy zapewnić wielozadaniowość.
- Powtórzenie wiadomości o transmisji szeregowej UART.

3. Omówienie przez nauczyciela elementów programowania obiektowego.

Programowanie obiektowe to operowanie w programie komputerowym zdefiniowanymi wcześniej wzorcami i tworzenie na ich podstawie elementów zwanych **obiektami**. Obiekty tworzone są na podstawie wzorca zwanego **klasą**.

obiekt
klasa

W klasie możemy zdefiniować m.in. dane zwane również **atrybutami** oraz **metody**, czyli funkcje operujące na obiekcie klasy.

atrybut
metoda

W klasie można dokonać definicji danych i metod jako publiczne lub prywatne. Publiczne to takie dane (atrybuty) i metody, do których możemy mieć dostęp z zewnątrz obiektu. W przypadku danych i metod prywatnych możemy się do nich odwołać tylko z wnętrza klasy. Czyli np. tylko metody tej klasy mogą operować na danych i metodach prywatnych. Jeśli chcemy zmieniać jakąś daną prywatną, to musimy do niej utworzyć metodę publiczną, której przekażemy dane do zmiany atrybutów prywatnych.

Przykładowa definicja klasy:

```
class ⟨Nazwa⟩{
private:
        // deklaracja danych i metod prywatnych

public:
        // deklaracja danych i metod publicznych

}
```

Ponadto w klasie mogą znajdować się tzw. konstruktory klasy. Są to metody, które wykonywane są przez program w momencie utworzenia obiektu danej klasy. Aby zdefiniować konstruktor klasy, musimy w klasie zbudować metodę o nazwie dokładnie takiej jak nazwa naszej klasy.

4. Ćwiczenie 1 Do realizacji wraz z nauczycielem.

Napisz program, który wyświetli trzy znaki `char` w losowych miejscach na wyświetlaczu LCD.

Znak powinien być obiektem, klasy `Znak`.

Klasa `Znak` powinna zawierać następujące dane i metody publiczne:

- `char znak; // pole przechowujące symbol do wyświetlenia`
- `int h; // współrzędna pozioma`
- `int v; // współrzędna pionowa`
- `void wyswietl(); // metoda wyświetlająca znak na wyświetlaczu.`

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// definicja klasy Znak
class Znak
{
  public:// definicja pól i metod publicznych
    char znak; // pole przechowujące symbol do wyświetlenia
    int h;// współrzędna pozioma
    int v;// współrzędna pionowa
    void wyswietl();// nagłówek metody wyświetlającej znak na wyświetlaczu.
};

// definicja metody wyświetlającej znak.
void Znak::wyswietl()
{
    lcd.setCursor(h, v);
    lcd.print(znak);
}

void setup() {
  // ustawienie ”ziarna” liczb losowych
  randomSeed(analogRead(0));
  lcd.begin(16, 2);
  delay(200);
  lcd.clear();

  // utworzenie obiektów kasy ”Znak”
  Znak znak1;
  Znak znak2;
  Znak znak3;

  // przypisanie do obiektów znaków char
  znak1.znak='X';
  znak2.znak='Y';
```

```
  znak3.znak='Z';

  // przypisanie dla każdego obiektu losowych wartości pól/współrzędnych
  znak1.h=random(16);
  znak1.v=random(2);
  znak2.h=random(16);
  znak2.v=random(2);
  znak3.h=random(16);
  znak3.v=random(2);

  // wywołanie metody wyswietl dla trzech obiektów
  znak1.wyswietl();
  znak2.wyswietl();
  znak3.wyswietl();
}

void loop() {
}
```

5. Ćwiczenie 2 Do realizacji wraz z nauczycielem.

Zmodyfikuj program z ćwiczenia 1 tak, by pola `znak`, `h` i `v` były polami prywatnymi klasy. Pola `h` i `v` powinny być definiowane losowo za pomocą konstruktora klasy. Polu `znak` powinna zostać przypisana wartość podana w parametrze konstruktora.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// definicja klasy Znak
class Znak
{
  public:// definicja pól i metod publicznych
    znak(char parametr);// nagłówek konstruktora z parametrem klasy Znak.
    void wyswietl();// nagłówek metody wyświetlającej znak na wyświetlaczu.

  private:
    char znak; // pole przechowujące symbol do wyświetlenia
    int h;// współrzędna pozioma
    int v;// współrzędna pionowa
};

// definicja metody wyświetlającej znak.
void znak::wyswietl()
{
  lcd.setCursor(h, v);
```

```
  lcd.print(znak);
}

// definicja konstruktora klasy
Znak::znak(char parametr)
{ // przypisanie dla każdego obiektu losowych wartości pól/współrzędnych
  h=random(16);
  v=random(2);

  // przypisanie znaku na padstawie parametru przy tworzeniu obiektu.
  znak=parametr;
}
void setup() {
  // ustawienie ziarna liczb losowych
  randomSeed(analogRead(0));
  lcd.begin(16, 2);
  delay(200);
  lcd.clear();

  // utworzenie obiektów klasy Znak
  Znak znak1('X');
  Znak znak2('Y');
  Znak znak3('Z');

  // wywołanie metody wyswietl dla trzech obiektów
  znak1.wyswietl();
  znak2.wyswietl();
  znak3.wyswietl();
}

void loop() {
}
```

6. Wprowadzenie przez nauczyciela funkcji wyświetlającej linie i prostokąty na wyświetlaczu OLED

Pracując z biblioteką `Adafruit_GFX.h` możemy możemy korzystać z jej bogatej oferty funkcji rysujących na wyświetlaczu OLED. Kilka z nich może być nam przydatne do ćwiczeń, które wykonamy w trakcie tej lekcji.

— Funkcja rysująca linię prostą od punktu o współrzędnych (x0,y0) do punktu (x1,y1):
`void drawLine(uint16_t x0,uint16_t y0,uint16_t x1,uint16_t y1,uint16_t color);`

— Funkcja rysująca obwód prostokąta od punktu o współrzędnych (x0,y0) – pierwszy róg prostokąta – do punktu (x1,y1) – przeciwległy róg prostokąta:
`void drawRect(uint16_t x0,uint16_t y0,uint16_t w,uint16_t h,uint16_t color);`

— Funkcja rysująca wypełniony kolorem prostokąt od punktu o współrzędnych (x0,y0)-pierwszy róg prostokąta do punktu (x1,y1) – przeciwległy róg prostokąta:

```
void fillRect(uint16_t x0,uint16_t y0,uint16_t w,uint16_t h,uint16_t color);
```

- Funkcja rysująca okrąg o środku w punkcie (x0,y0) i o promieniu r:
  `void drawCircle(uint16_t x0,uint16_t y0,uint16_t r,uint16_t color);`
- Funkcja rysująca koło wypełnione kolorem o środku w punkcie (x0,y0) i o promieniu r:
  `void fillCircle(uint16_t x0,uint16_t y0,uint16_t r,uint16_t color);`

W naszych programach ze względu na rodzaj wyświetlacza stosujemy kolor biały (`WHITE`).

7. Ćwiczenie 3 Do realizacji samodzielnej uczniów.

Napisz program, który wyświetli na wyświetlaczu OLED kwadrat o rozmiarze 2×2 px.
Kwadrat powinien być umiejscowiony na wyświetlaczu OLED w jego centralnym punkcie.
Bok kwadratu powinien być określany za pomocą ciągłych odczytów wartości potencjometru.

Kwadrat powinien być obiektem klasy „kwadrat".

Klasa „kwadrat" powinna zawierać następujące dane i metody publiczne:

- `kwadrat(); // konstruktor definiujący parametry początkowe.`
- `void zmien_bok(int rozmiar); // nagłówek metody zmieniającej rozmiar boku kwadratu`
- `void wyswietl(); // metoda wyświetlająca kwadrat na wyświetlaczu OLED.`

Oraz pola prywatne:

- `int bok; // bok kwadratu`
- `int h; // współrzędna pozioma`
- `int v; // współrzędna pionowa`

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);
// definicja klasy kwadrat
class kwadrat
{
  public:// definicja pól i metod publicznych
    kwadrat();// nagłówek konstruktora bez parametru.
    void wyswietl();// nagłówek metody wyświetlającej znak na wyświetlaczu.
    void zmien_bok(int rozmiar); // nagłówek metody w zmieniającej wartość rozmiaru boku kwadratu

  private:// definicja pól prywatnych klasy
```

```
    int bok;// bok kwadratu
    int h;// współrzędna pozioma srodka
    int v;// współrzędna pionowa srodka
};

void kwadrat::zmien_bok(int rozmiar)
{
  bok=rozmiar;
}
// definicja metody wyświetlającej kwadrat.
void kwadrat::wyswietl()
{
  delay(100);
  // wyczyszczenie ekranu OLED
  display.clearDisplay();

  // obliczenie położenia lewego dolnego rogu kwadratu
  uint16_t x0=h-0.5*bok;
  uint16_t y0=v-0.5*bok;
  delay(100);
  // zastosowanie funkcji rysującej prostokąt
  display.drawRect(x0, y0, bok, bok, 1);

  // wyświetlenie zdefiniowanych obiektów na wyświetlaczy OLED
  display.display();
}

// definicja konstruktora klasy
kwadrat::kwadrat()
{
  // wywołanie metody przypisującej rozmiar boku
  zmien_bok(1);
  // przypisanie współrzędnym h i v środkowej pozycji ekranu OLED
  h=64;
  v=32;
}

// utworzenie obiektu kwadrat1 klasy kwadrat
kwadrat kwadrat1;

void setup() {
  display.begin(0x2, 0x3C, false);
  delay(200);
}

void loop() {
  // odczyt i przeskalowanie danej odczytanej z potencjometru
  // wywołanie metody ustawiającej wartość boku
  kwadrat1.zmien_bok(analogRead(A1)/(1023/63));
```

```
  // wywołanie metody wyświetlającej kwadrat;
  kwadrat1.wyswietl();
}
```

8. Ćwiczenie 4 Do realizacji samodzielnej przez uczniów.

Na podstawie ćwiczenia 3 zdefiniuj tablicę obiektów `kwadrat` o rozmiarze N, gdzie N to wielkość zdefiniowana w stałej programowej.

Zmodyfikuj konstruktor klasy `kwadrat` tak, by wyznaczał on losowe wartości współrzędnych środka dla prostokątów.

Długość boku prostokąta nadal powinna być odczytywana z wejścia analogowego A1, tj. potencjometru.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

// definicja rozmiaru tablicy kwadrat
#define N 5

Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);
// definicja klasy kwadrat
class kwadrat
{
  public: // definicja pól i metod publicznych
    kwadrat();// nagłówek konstruktora bez parametru.
    void wyswietl();// nagłówek metody wyświetlającej znak na wyświetlaczu.
    void zmien_bok(int rozmiar); // nagłówek metody zmieniającej długość boku kwadratu
    void losuj_srodek();

  private: // definicja pól prywatnych klasy
    int bok;// bok kwadratu
    int h;// współrzędna pozioma srodka
    int v;// współrzędna pionowa srodka
};

void kwadrat::zmien_bok(int rozmiar)
{
  bok=rozmiar;
}
// definicja metody wyświetlającej kwadrat.
```

```
void kwadrat::wyswietl()
{
  // obliczenie położenia lewego dolnego rogu kwadratu
  uint16_t x0=h-0.5*bok;
  uint16_t y0=v-0.5*bok;
  // delay(10);
  // zastosowanie funkcji rysującej prostokąt
  display.drawRect(x0, y0, bok, bok, 1);

  // wyświetlenie zdefiniowanych obiektów na wyświetlaczy OLED
  display.display();
}

// definicja konstruktora klasy
kwadrat::kwadrat()
{
  // wywołanie metody przypisującej rozmiar boku
  zmien_bok(1);

  // wywołanie metody losującej współrzędne środka
  losuj_srodek();
}

void kwadrat::losuj_srodek()
{
    // przypisanie współrzędnym h i v środkowej pozycji ekranu OLED
  h=random(128);
  v=random(64);
}

// utworzenie obiektu kwadrat1 klasy kwadrat
kwadrat kwadraty[N];

void setup() {
  randomSeed(analogRead(0));
  for(int i=0; i<N;i++)
  {
    kwadraty[i].losuj_srodek();
  }
  display.begin(0x2, 0x3C, false);
  delay(200);
}

void loop() {
  // delay(10);
  // wyczyszczenie ekranu OLED
  display.clearDisplay();
  // pętla iterująca tablicę obiektów kwadraty
```

```
  for(int i=0; i<N;i++)
  {
    // odczyt i przeskalowanie danej odczytanej z potencjometru
    // wywołanie metody ustawiającej wartość boku
    kwadraty[i].zmien_bok(analogRead(A1)/(1023/63));
    // wywołanie metody wyświetlającej kwadrat;
    kwadraty[i].wyswietl();
  }
}
```

9. Ćwiczenie 5 Do realizacji w grupie uczniowskiej.

Napisz program, który wyświetli losowo wypełnioną planszę gry w „Kółko i krzyżyk".

Plansza gry w „Kółko i krzyżyk" powinna być klasą `kolko_i_krzyzyk`.

W klasie tej powinniśmy zdefiniować minimalnie następujące dane i metody.

Prywatne:

```
int plansza[3][3]; // tablica przechowująca wartości poszczególnych pól planszy gry
void siatka();// nagłówek metody wyświetlającej linie siatki gry
void rysuj_kolko();
void rysoj_krzyzyk();
void losuj_plansze();
```

Publiczne:

```
void wyswietl_gre();//definicja nagłówka metody wyświetlającej
wypełnioną planszę gry
kolko_i_krzyzyk(); //definicja nagłówka konstruktora klasy
```

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

// definicja rozmiaru tablicy kwadrat
#define N 5

Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);

class kolko_i_krzyzyk
{
  private:
```

```
    int plansza[3][3]; // tablica przechowująca wartości poszczególnych pól planszy gry
    void siatka();// nagłówek metody wyświetlającej linie siatki gry
    void rysuj_kolko(int x, int y);
    void rysuj_krzyzyk(int x, int y);
    void losuj_plansze();
  public:
    void wyswietl_gre();// definicja nagłówka metody wyświetlającej wypełnioną planszę gry
    kolko_i_krzyzyk(); // definicja nagłówka konstruktora klasy

};

void kolko_i_krzyzyk::siatka()
{
  for(int x0=128/3; x0<(128/3)*2+1; x0+=128/3)
  {
     // zastosowanie funkcji rysującej linię pionową
    display.drawLine(x0, 0, x0, 64, 1);
  }
  for(int y0=64/3; y0<(64/3)*2+1; y0+=64/3)
  {
     // zastosowanie funkcji rysującej linię poziomą
    display.drawLine(0, y0, 128, y0, 1);
  }
}

void kolko_i_krzyzyk::rysuj_kolko(int x, int y)
{
  display.drawCircle(x, y, 5, 1);
}
void kolko_i_krzyzyk::rysuj_krzyzyk(int x, int y)
{
  // narysowanie krzyżyka składającgo się z dwóch linii
  display.drawLine(x-5,y-5,x+5,y+5,1);
  display.drawLine(x-5,y+5,x+5,y-5,1);
}

void kolko_i_krzyzyk::losuj_plansze()
{
  for(int i=0; i<3; i++)
    for(int j=0; j<3;j++)
     {
        plansza[i][j]=random(2);
     }
}

void kolko_i_krzyzyk::wyswietl_gre()
{
  // wyczyszczenie ekranu OLED
```

```
  display.clearDisplay();
  siatka();
  for(int i=0; i<3; i++)
    for(int j=0; j<3;j++)
     {
        if(plansza[i][j]==0)
          rysuj_kolko(128/6+i*128/3,64/6+j*64/3);
        else
          rysuj_krzyzyk(128/6+i*128/3,64/6+j*64/3);
     }
  display.display();
}

kolko_i_krzyzyk::kolko_i_krzyzyk()
{
  losuj_plansze();
}

void setup() {
  randomSeed(analogRead(0));
  kolko_i_krzyzyk gra;
  display.begin(0x2, 0x3C, false);
  delay(200);
  gra.wyswietl_gre();
}

void loop() {
}
```

10. Ćwiczenie 6 Do realizacji w grupie uczniowskiej.

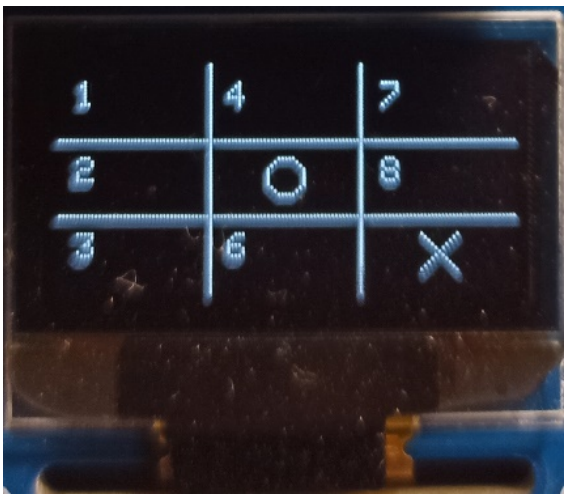
Na podstawie programu napisanego w ćwiczeniu 5 napisz program, w którym przeprowadzisz grę w „Kółko i krzyżyk" układu ArduinoUNO z użytkownikiem.
Kolejne kroki rozgrywki powinieneś wyświetlać na wyświetlaczu OLED.
Użytkownik wykonuje ruch podając pozycję dla kolejnego symbolu w postaci cyfry od 1 do 9, przesyłanej po UART.

Jeśli pole jest już zajęte, układ Arduino powinien czekać na ponowny (inny) ruch użytkownika.

Ruch ArduinoUNO powinien następować jako drugi po użytkowniku. Powinien być to ruch losowy na jedno z pól jeszcze nie zajętych.

Widok przykładowej siatki planszy na początku rozgrywki:

Widok przykładowej planszy po kolejnych ruchach graczy:



Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
// wczytanie bibliotek obsługujących wyświetlacz OLED
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <Wire.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define RESET_BTN 4

Adafruit_SSD1306 display(SCREEN_WIDTH,SCREEN_HEIGHT, &Wire,
RESET_BTN);

int liczba=0;

class kolko_i_krzyzyk
{
  private:
    int plansza[3][3]; // tablica przechowująca wartości poszczególnych pól planszy gry
```

```
    void siatka();// nagłówek metody wyświetlającej linie siatki gry
    void rysuj_kolko(int x, int y);// nagłówek metody rysującej kółko
    void rysuj_krzyzyk(int x, int y);// nagłówek metody rysującej krzyżyk
  public:
    bool ruch_uzytkownika(int cyfra);// //nagłówek metody zaznaczającej ruch użytkownika
    bool ruch_komputera(); // nagłówek metody losującej ruch ArduinoUNO
    void wyswietl_gre();// definicja nagłówka metody wyświetlającej wypełnioną planszę gry
    kolko_i_krzyzyk(); // definicja nagłówka konstruktora klasy
    void losuj_plansze();// nagłówek metody losującej planszę
};

void kolko_i_krzyzyk::siatka()
{
  for(int x0=128/3; x0<(128/3)*2+1; x0+=128/3)
  {
     // zastosowanie funkcji rysującej linię pionową
    display.drawLine(x0, 0, x0, 64, 1);
  }
  for(int y0=64/3; y0<(64/3)*2+1; y0+=64/3)
  {
     // zastosowanie funkcji rysującej linię poziomą
    display.drawLine(0, y0, 128, y0, 1);
  }
}

void kolko_i_krzyzyk::rysuj_kolko(int x, int y)
{
  // wyświetlenie okręgu o środku we współrzędnych x, y i promieniu 5
  display.drawCircle(x, y, 5, 1);
}
void kolko_i_krzyzyk::rysuj_krzyzyk(int x, int y)
{
  // narysowanie krzyżyka składającgo się z dwóch linii
  display.drawLine(x-5,y-5,x+5,y+5,1);
  display.drawLine(x-5,y+5,x+5,y-5,1);
}

void kolko_i_krzyzyk::losuj_plansze()
{
  // wypełnienie tablicy losowymi wartościami 0, 1
  for(int i=0; i<3; i++)
    for(int j=0; j<3;j++)
     {
        plansza[i][j]=random(2);
     }
}
```

```
void kolko_i_krzyzyk::wyswietl_gre()
{
  // wyczyszczenie ekranu OLED
  display.clearDisplay();
  siatka();// wyświetlenie siatki planszy
  for(int i=0; i<3; i++)
    for(int j=0; j<3;j++)
     {
        // instrukcja warunkowa wyświetlająca ruchy użytkownika na planszy
        if(plansza[i][j]==0)
          rysuj_kolko(128/6+i*128/3,64/6+j*64/3);
        // instrukcja warunkowa wyświetlająca ruchy układu ArduinoUNO na planszy
        if(plansza[i][j]==1)
          rysuj_krzyzyk(128/6+i*128/3,64/6+j*64/3);
        // instrukcja warunkowa wyświetlająca cyfry pod numerami pól w których
        // nie zostało jeszcze postawione kółko lub krzyżyk
        if(plansza[i][j]==999)
        {
          display.setTextColor(WHITE);
          display.setCursor(5+i*128/3,5+j*64/3);
          display.println(i*3+j+1);
        }
     }
  display.display();
}

bool kolko_i_krzyzyk::ruch_uzytkownika(int cyfra)
{
  // instrukcja warunkowa sprawdzający, czy we wskazanym przez użytkownika miejscu
  // na planszy nie odnotowano już innego ruchu
  if(plansza[(cyfra-1)/3][(cyfra-1)%3]==1||plansza[(cyfra-1)/3][(cyfra-1)%3]==0)
  {
    liczba=0;
    return 0;
  }
  else
    plansza[(cyfra-1)/3][(cyfra-1)%3]=0;
  return 1;
}

bool kolko_i_krzyzyk::ruch_komputera()
{
  int i,j;

  // losowanie współrzędnych i,j do momentu kiedy współrzędne wskażą
  // miejsce w tablicy gry w którym jeszcze nie odnotowano ruchu graczy
  do{
```

```
    i=random(3);
    j=random(3);
  }while(plansza[i][j]==1||plansza[i][j]==0);
  plansza[i][j]=1;

  return 1;
}

// konstruktor klasy kolko_i_krzyzyk który inicjuje wartosci w tablicy gry plansza[][]
kolko_i_krzyzyk::kolko_i_krzyzyk()
{
  for(int i=0; i<3; i++)
    for(int j=0; j<3;j++)
      plansza[i][j]=999;
}

// deklaracja obiektu gra klasy kolko_i_krzyzyk
kolko_i_krzyzyk gra;

void setup() {
  randomSeed(analogRead(0));// ustaweinie ziarna liczb losowych
  display.begin(0x2, 0x3C, false);// włączenie wyświwtlacza OLED
  delay(200);
  gra.wyswietl_gre();// wyświetlenie aktualnego stanu gry(tu tylko siatki)
  // włączamy odbiornik podczerwieni
  Serial.begin(9600);
  delay(2000);
}

void loop() {
  // wykonujemy odczyt cyfr z UART do momentu,
  // kiedy podana liczba będzie wskazywała wolne pole gry.
  // metoda gra.ruch_uzytkownika(liczba) zapisze ruch gracza i zwróci wtedy wartość 1
  do
  {
    while(liczba==0) { // Czy odebrano jakiś tekst?
      if(Serial.available() > 0)
        liczba = Serial.parseInt();
    }
    delay(200);
    Serial.flush();
  }while(!gra.ruch_uzytkownika(liczba)); // znak '!' to negacja wartości
                                         // zwracanej przez metodę ruch_uzytkownika()

  liczba=0;
  gra.wyswietl_gre();// wyświetlenie aktualnego stanu gry
  delay(2000);
  // losujemy ruch układu ArduinoUNO
```

```
        gra.ruch_komputera();
        gra.wyswietl_gre();
        delay(2000);
    }
```

11. Przeprowadzenie quizu podsumowującego lekcję.

1. Programowanie obiektowe to:
   a) To samo, co programowanie wizualne
   b) Programowanie za pomocą instrukcji ze specjalnych bibliotek
   c) Programowanie na podstawie wzorców i tworzenie na ich podstawie tzw. obiektów.
2. Jeśli w klasie określimy jakiś atrybut jako prywatny to:
   a) Nie będziemy mieli żadnej możliwości edycji tego atrybutu.
   b) Będzie możliwa edycja za pośrednictwem metody w tej samej klasie.
   c) Jako właściciele programu będziemy mogli zawsze edytować ten atrybut.
   d) Dla edycji nie ma znaczenia czy atrybut jest publiczny, czy prywatny
3. Metodą klasy w programowaniu obiektowym nazwiemy:
   a) Sposób grupy uczniów na rozwiązanie jakiegoś problemu algorytmicznego.
   b) Dane zapisane w klasie.
   c) Funkcje, które działają wewnątrz obiektu klasy i są w niej zdefiniowane.
   d) Język programowania w którym piszemy program.
4. Konstruktor klasy to:
   a) Programista, który zbudował program.
   b) Każda dana/atrybut w klasie.
   c) Każda metoda w klasie.
   d) Metoda wykonywana przy tworzeniu obiektu tej klasy.

Poprawne odpowiedzi zostały zaznaczone.

Karta ćwiczeń

Ćwiczenie 1

Napisz program, który wyświetli trzy znaki `char` w losowych miejscach na wyświetlaczu LCD.

Znak powinien być obiektem, klasy `Znak`.

Klasa `Znak` powinna zawierać następujące dane i metody publiczne:

- `char znak; // pole przechowujące symbol do wyświetlenia`
- `int h; // współrzędna pozioma`
- `int v; // współrzędna pionowa`
- `void wyswietl(); // metoda wyświetlająca znak na wyświetlaczu.`

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// definicja klasy Znak
class Znak
{
  public:// definicja pól i metod publicznych
    char znak; // pole przechowujące symbol do wyświetlenia
    int h;// współrzędna pozioma
    int v;// współrzędna pionowa
    void wyswietl();// nagłówek metody wyświetlającej znak na wyświetlaczu.
};

// definicja metody wyświetlającej znak.
void Znak::wyswietl()
{
    lcd.setCursor(h, v);
    lcd.print(znak);
}

void setup() {
  // ustawienie "ziarna" liczb losowych
  randomSeed(analogRead(0));
  lcd.begin(16, 2);
```

```
  delay(200);
  lcd.clear();

  // utworzenie obiektów kasy "Znak"
  Znak znak1;
  Znak znak2;
  Znak znak3;

  // przypisanie do obiektów znaków char
  znak1.znak='X';
  znak2.znak='Y';
  znak3.znak='Z';

  // przypisanie dla każdego obiektu losowych wartości pól/współrzędnych
  znak1.h=random(16);
  znak1.v=random(2);
  znak2.h=random(16);
  znak2.v=random(2);
  znak3.h=random(16);
  znak3.v=random(2);

  // wywołanie metody wyswietl dla trzech obiektów
  znak1.wyswietl();
  znak2.wyswietl();
  znak3.wyswietl();
}

void loop() {
}
```

Ćwiczenie 2

Zmodyfikuj program z ćwiczenia 1 tak, by pola `znak`, `h` i `v` były polami prywatnymi klasy. Pola `h` i `v` powinny być definiowane losowo za pomocą konstruktora klasy. Polu `znak` powinna zostać przypisana wartość podana w parametrze konstruktora.

Przykładowy kod źródłowy rozwiązania ćwiczenia:

```
#include <Wire.h>
#include <hd44780.h>
#include <hd44780ioClass/hd44780_I2Cexp.h>

// konfiguracja LCD
hd44780_I2Cexp lcd(0x20, I2Cexp_MCP23008,7,6,5,4,3,2,1,HIGH);

// definicja klasy Znak
class Znak
{
  public:// definicja pól i metod publicznych
    znak(char parametr);// nagłówek konstruktora z parametrem klasy Znak.
```

```
    void wyswietl();// nagłówek metody wyświetlającej znak na wyświetlaczu.

  private:
    char znak; // pole przechowujące symbol do wyświetlenia
    int h;// współrzędna pozioma
    int v;// współrzędna pionowa
};

// definicja metody wyświetlającej znak.
void znak::wyswietl()
{
  lcd.setCursor(h, v);
  lcd.print(znak);
}

// definicja konstruktora klasy
Znak::znak(char parametr)
{ // przypisanie dla każdego obiektu losowych wartości pól/współrzędnych
  h=random(16);
  v=random(2);

  // przypisanie znaku na padstawie parametru przy tworzeniu obiektu.
  znak=parametr;
}
void setup() {
  // ustawienie ziarna liczb losowych
  randomSeed(analogRead(0));
  lcd.begin(16, 2);
  delay(200);
  lcd.clear();

  // utworzenie obiektów klasy Znak
  Znak znak1('X');
  Znak znak2('Y');
  Znak znak3('Z');

  // wywołanie metody wyswietl dla trzech obiektów
  znak1.wyswietl();
  znak2.wyswietl();
  znak3.wyswietl();
}

void loop() {
}
```

Ćwiczenie 3

Napisz program, który wyświetli na wyświetlaczu OLED kwadrat o rozmiarze 2×2 px.
Kwadrat powinien być umiejscowiony na wyświetlaczu OLED w jego centralnym punkcie.
Bok kwadratu powinien być określany za pomocą ciągłych odczytów wartości potencjometru.

Kwadrat powinien być obiektem klasy „kwadrat".

Klasa „kwadrat" powinna zawierać następujące dane i metody publiczne:

- `kwadrat(); // konstruktor definiujący parametry początkowe.`
- `void zmien_bok(int rozmiar); // nagłówek metody zmieniającej rozmiar boku kwadratu`
- `void wyswietl(); // metoda wyświetlająca kwadrat na wyświetlaczu OLED.`

Oraz pola prywatne:

- `int bok; // bok kwadratu`
- `int h; // współrzędna pozioma`
- `int v; // współrzędna pionowa`

Ćwiczenie 4

Na podstawie ćwiczenia 3 zdefiniuj tablicę obiektów `kwadrat` o rozmiarze N, gdzie N to wielkość zdefiniowana w stałej programowej.

Zmodyfikuj konstruktor klasy `kwadrat` tak, by wyznaczał on losowe wartości współrzędnych środka dla prostokątów.

Długość boku prostokąta nadal powinna być odczytywana z wejścia analogowego A1, tj. potencjometru.

Ćwiczenie 5

Napisz program, który wyświetli losowo wypełnioną planszę gry w „Kółko i krzyżyk".

Plansza gry w „Kółko i krzyżyk" powinna być klasą `kolko_i_krzyzyk`.

W klasie tej powinniśmy zdefiniować minimalnie następujące dane i metody.

Prywatne:

```
int plansza[3][3]; // tablica przechowująca wartości poszczególnych pól planszy gry
void siatka();// nagłówek metody wyświetlającej linie siatki gry
void rysuj_kolko();
void rysoj_krzyzyk();
void losuj_plansze();
```

Publiczne:

```
void wyswietl_gre();//definicja nagłówka metody wyświetlającej
wypełnioną planszę gry
kolko_i_krzyzyk(); //definicja nagłówka konstruktora klasy
```

Ćwiczenie 6

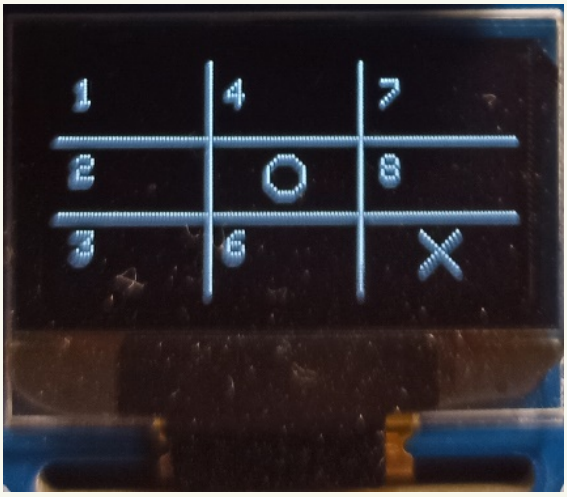
Na podstawie programu napisanego w ćwiczeniu 5 napisz program, w którym przeprowadzisz grę w „Kółko i krzyżyk” układu ArduinoUNO z użytkownikiem.
Kolejne kroki rozgrywki powinieneś wyświetlać na wyświetlaczu OLED.
Użytkownik wykonuje ruch podając pozycję dla kolejnego symbolu w postaci cyfry od 1 do 9, przesyłanej po UART.

Jeśli pole jest już zajęte, układ Arduino powinien czekać na ponowny (inny) ruch użytkownika.

Ruch ArduinoUNO powinien następować jako drugi po użytkowniku. Powinien być to ruch losowy na jedno z pól jeszcze nie zajętych.

Widok przykładowej siatki planszy na początku rozgrywki:

Widok przykładowej planszy po kolejnych ruchach graczy:



Quiz

1. Programowanie obiektowe to:
   a) To samo, co programowanie wizualne
   b) Programowanie za pomocą instrukcji ze specjalnych bibliotek
   c) Programowanie na podstawie wzorców i tworzenie na ich podstawie tzw. obiektów.
2. Jeśli w klasie określimy jakiś atrybut jako prywatny to:
   a) Nie będziemy mieli żadnej możliwości edycji tego atrybutu.
   b) Będzie możliwa edycja za pośrednictwem metody w tej samej klasie.
   c) Jako właściciele programu będziemy mogli zawsze edytować ten atrybut.
   d) Dla edycji nie ma znaczenia czy atrybut jest publiczny, czy prywatny
3. Metodą klasy w programowaniu obiektowym nazwiemy:
   a) Sposób grupy uczniów na rozwiązanie jakiegoś problemu algorytmicznego.
   b) Dane zapisane w klasie.
   c) Funkcje, które działają wewnątrz obiektu klasy i są w niej zdefiniowane.
   d) Język programowania w którym piszemy program.
4. Konstruktor klasy to:
   a) Programista, który zbudował program.
   b) Każda dana/atrybut w klasie.
   c) Każda metoda w klasie.
   d) Metoda wykonywana przy tworzeniu obiektu tej klasy.

SZANOWNI PAŃSTWO,

STARAMY SIĘ DOKŁADAĆ WSZELKICH STARAŃ, ABY WSPIERAĆ POTRZEBY PAŃSTWA I UCZNIÓW

Niniejsza publikacja stanowi wsparcie merytoryczne dla nauczyciela w pracy z mikrokontrolerem Arduino Uno z zestawem edukacyjnym TME-EDU-ARD-2

Zachęcamy do kontaktu z przedstawicielami firmy Moje Bambino.
Nasi specjaliści chętnie pomogą, służąc swoją wiedzą i doświadczeniem.
Zaproś nas do swojej placówki lub umów się na konsultacje.

- Pomożemy w doborze sprzętu
- Zapewnimy bezpłatne konsultacje
- Wspólnie wybierzemy najkorzystniejsze rozwiązanie

ZACHĘCAMY DO KONTAKTU Z PRZEDSTAWICIELAMI FIRMY MOJE BAMBINO

Infolinia: (Pn-Pt 8:00-16:00)
- 0 801 577 544
- 42 630 01 30,
- 630 01 70,
- 630 01 34,
- 630 03 05

Faks: 42 630 04 80

e-mail: biuro@mojebambino.pl
e-mail: zamowienia@mojebambino.pl

Laboratoria Przyszłości:

Infolinia: 42 233 78 78
e-mail: laboratoriaprzyszlosci@mojebambino.pl

https://mojebambino.pl/przedstawiciele