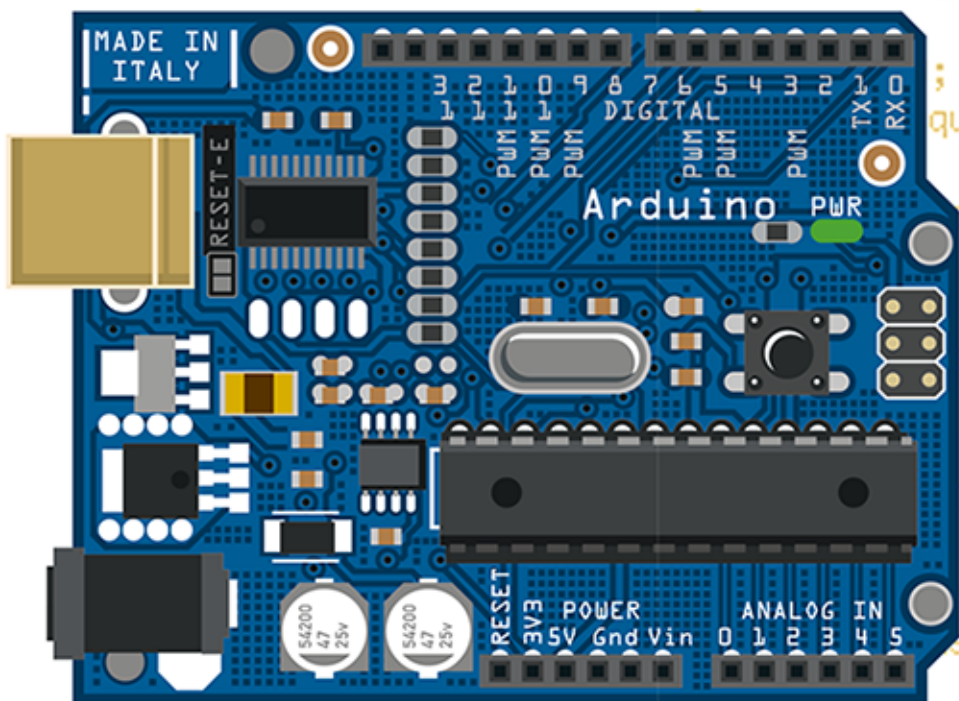




# Arduino dla początkujących Podstawy i szkice

Podręcznik programisty Arduino!



Simon Monk



Tytuł oryginału: Programming Arduino Getting Started with Sketches  
Tłumaczenie: Konrad Matuk

ISBN: 978-83-246-8707-7

Original edition copyright © 2012 by The McGraw-Hill Companies.  
All rights reserved.

Polish edition copyright © 2014 by HELION S.A.  
All rights reserved.

“Arduino” is a trademark of the Arduino team.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/ardupo>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

O autorze .....	9
Podziękowania .....	10
Wstęp .....	11
Czym jest Arduino? .....	11
Co będzie potrzebne? .....	12
Korzystanie z niniejszej książki .....	12
Pomoce .....	13
Rozdział 1. Oto Arduino .....	15
Mikrokontrolery .....	15
Płyty rozwojowe .....	16
Płyta Arduino .....	17
Zasilanie .....	17
Złącza zasilania .....	17
Wejścia analogowe .....	18
Złącza cyfrowe .....	18
Mikrokontroler .....	18
Pozostałe podzespoły .....	19
Początki Arduino .....	19
Rodzina płyt Arduino .....	21
Uno, Duemilanove i Diecimila .....	21
Mega .....	22
Nano .....	22
Bluetooth .....	23
Lilypad .....	24
Inne „oficjalne” płytki .....	24
Inne klony i odmiany Arduino .....	25
Podsumowanie .....	25

Rozdział 2. Rozpoczynamy przygodę z Arduino .....	27
Zasilanie .....	27
Instalacja oprogramowania .....	28
Ładowanie pierwszego szkicu .....	28
Aplikacja Arduino .....	33
Podsumowanie .....	34
 Rozdział 3. Podstawy języka C .....	35
Programowanie .....	35
Czym jest język programowania? .....	36
Blink po raz kolejny .....	40
Zmienne .....	42
Eksperymentowanie w języku C .....	44
<i>Zmienne numeryczne i arytmetyka</i> .....	45
Polecenia .....	47
<i>if</i> .....	47
<i>for</i> .....	49
<i>while</i> .....	51
Dyrektywa <i>#define</i> .....	52
Podsumowanie .....	52
 Rozdział 4. Funkcje .....	53
Czym jest funkcja? .....	53
Parametry .....	54
Zmienne globalne, lokalne i statyczne .....	55
Zwracanie wartości .....	58
Zmienne innych typów .....	59
<i>float</i> .....	59
<i>boolean</i> .....	60
<i>Inne typy danych</i> .....	61
Styl zapisu kodu .....	62
<i>Wcięcia</i> .....	62
<i>Nawiasy klamrowe otwierające</i> .....	63
<i>Białe znaki</i> .....	63
<i>Komentarze</i> .....	64
Podsumowanie .....	65
 Rozdział 5. Tablice i łańcuchy .....	67
Tablice .....	67
<i>Zastosowanie tablic do alfabetu Morse'a i sygnału SOS</i> .....	70
Tablice łańcuchów .....	71
<i>Literały łańcuchowe</i> .....	71
<i>Zmienne łańcuchowe</i> .....	72
Tłumacz alfabetu Morse'a .....	73
<i>Dane</i> .....	73
<i>Zmienne globalne i funkcja setup</i> .....	74

<i>Funkcja loop</i> .....	75
<i>Funkcja flashSequence</i> .....	77
<i>Funkcja flashDotOrDash</i> .....	78
<i>Składanie całości programu</i> .....	78
Podsumowanie .....	80
Rozdział 6. Wejścia i wyjścia .....	81
Wyjścia cyfrowe .....	81
Wejścia cyfrowe .....	84
<i>Rezystor podwyższający</i> .....	85
<i>Wewnętrzny rezystor podwyższający</i> .....	88
<i>Usuwanie stuków</i> .....	88
Wyjścia analogowe .....	93
Wejścia analogowe .....	95
Podsumowanie .....	96
Rozdział 7. Standardowa biblioteka Arduino .....	97
Liczby losowe .....	97
Funkcje matematyczne .....	99
Manipulacja bitami .....	99
Zaawansowane funkcje wejścia i wyjścia .....	102
<i>Generowanie tonów</i> .....	102
<i>Wprowadzanie rejestru przesuwnego</i> .....	103
Przerwania .....	103
Podsumowanie .....	105
Rozdział 8. Zapisywanie danych .....	107
Stałe .....	107
Dyrektywa PROGMEM .....	108
EEPROM .....	109
<i>Przechowywanie wartości zmiennej typu int w pamięci EEPROM</i> .....	110
<i>Przechowywanie wartości typu float w pamięci EEPROM (unie)</i> .....	110
<i>Przechowywanie łańcucha w pamięci EEPROM</i> .....	111
<i>Wymazywanie zawartości pamięci EEPROM</i> .....	112
Kompresja .....	112
<i>Kompresja zakresu</i> .....	112
Podsumowanie .....	113
Rozdział 9. Wyświetlacze LCD .....	115
Tablica wyświetlająca komunikaty za pośrednictwem interfejsu USB .....	116
Korzystanie z wyświetlacza .....	118
Inne funkcje biblioteki wyświetlacza LCD .....	119
Podsumowanie .....	120

Rozdział 10. Programowanie aplikacji sieci Ethernet .....	121
Płytki pozwalające na pracę w sieci Ethernet .....	122
Komunikacja z serwerami sieciowymi .....	122
HTTP .....	122
HTML .....	122
Arduino w roli serwera sieci Web .....	123
Konfigurowanie złączy Arduino za pośrednictwem sieci .....	126
Podsumowanie .....	131
Rozdział 11. C++ i biblioteki .....	133
Mechanizmy obiektowe .....	133
Klasy i metody .....	133
Przykład wbudowanej biblioteki .....	134
Tworzenie bibliotek .....	134
Plik nagłówkowy .....	134
Plik implementacji .....	136
Uzupełnianie swojej biblioteki .....	137
Podsumowanie .....	139
Skorowidz .....	141

## Rozdział 5.

# Tablice i łańcuchy

Po lekturze rozdziału 4. już wiesz, jak należy dzielić szkic tak, aby łatwiej się z nim pracowało. Każdy dobry programista lubi, gdy praca przebiega łatwo i sprawnie. Teraz przyjrzymy się strukturom danych, które będziesz stosować w swoich szkicach.

Książka pod tytułem *Algorytmy + struktury danych = programy* napisana przez Niklause Wirtha została wydana już jakiś czas temu, jednakże wciąż dobrze opisuje sedno informatyki, a zwłaszcza programowania. Tę książkę mogę polecić każdej osobie zmagającej się z błędami w tworzonych programach. Książka opisuje ideę, w myśl której pisząc program, należy myśleć zarówno o algorytmie (przeprowadzanych operacjach), jak również o stosowanych strukturach danych.

Przeanalizowałeś już pewne funkcje, pętle i instrukcje, czyli coś, co można określić mianem „algorytmicznej” strony programowania Arduino. Teraz przeanalizujemy różne struktury danych.

---

## Tablice

Tablice służą do przechowywania zbioru zmiennych. Zmienne, które do tej pory poznałeś, przechowywały tylko jedną wartość. Zwykle była to wartość typu `int`. Tablica natomiast zawiera listę zmiennych. Korzystając z tej listy, możesz uzyskać dostęp do dowolnej wartości zapisanej w tablicy.

W języku C, tak jak w większości języków programowania, numeracja indeksów zaczyna się od 0, a nie od 1. A więc pierwszym elementem tablicy jest element o numerze zero.

W celu zademonstrowania działania tablic stworzymy przykładową aplikację, która za pomocą błysków wbudowanej w Arduino diody LED będzie generować alfabetem Morse’a sygnał „SOS”.

Alfabet Morse’a był stosowany do komunikacji w XIX i XX wieku. Składa się z kombinacji dwóch znaków — kropki i kreski. Dzięki temu mógł on być przesyłany za pośrednictwem kabli telegraficznych i łączy radiowych. Kod ten również mógł być przekazywany

przy pomocy sygnałów świetlnych. Skrót „SOS” (od ang. *save our souls*) wciąż odgrywa rolę międzynarodowego sygnału oznaczającego wołanie o pomoc.

Litera *S* jest reprezentowana przez trzy krótkie błyski (kropki), a litera *O* — przez trzy długie błyski (kreski). Utworzymy tablicę, która będzie przechowywać elementy typu `int` zawierające informacje o długości każdego błysku. Elementy tablicy zostaną następnie zastosowane w pętli `for` w celu określenia odpowiedniego czasu trwania błysków.

Najpierw przyjrzyjmy się sposobowi tworzenia tablicy elementów typu `int`, które mają określać czas trwania poszczególnych błysków.

```
int durations[] = {200, 200, 200, 500, 500, 500, 200, 200, 200};
```

Umieszczając nawiasy kwadratowe `[]` za nazwą zmiennej, wskazujemy, że zmienna ta zawiera tablicę.

W zaprezentowanym przykładzie określamy wartości długości błysków w momencie tworzenia tablicy. Deklaracja tych elementów polega na zastosowaniu nawiasów klamrowych, a następnie wpisaniu wartości oddzielonych od siebie przecinkami. Nie zapomnij o umieszczeniu średnika na końcu linii kodu.

Dostęp do dowolnego elementu tablicy możesz uzyskać przy użyciu zapisu z nawiasem kwadratowym. Jeżeli chcesz uzyskać dostęp do pierwszego elementu tablicy, zastosuj poniższy kod:

```
durations[0]
```

Aby zademonstrować to w praktyce, stwórzmy tablicę, a następnie wyświetlimy wartości zapisane w tej tablicy przy użyciu monitora portu szeregowego:

```
//szkic 05.01.
int ledPin = 13;

int durations[] = {200, 200, 200, 500, 500, 500, 200, 200, 200};

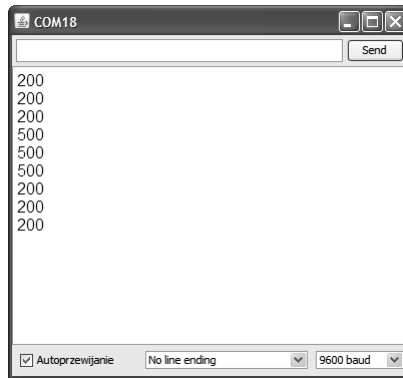
void setup()
{
  Serial.begin(9600);
  for (int i = 0; i < 9; i++)
  {
    Serial.println(durations[i]);
  }
}

void loop() {}
```

Włóż skrzynkę do Arduino, a następnie otwórz monitor portu szeregowego. Efekt działania programu powinien być podobny do tego, co przedstawiono na rysunku 5.1.

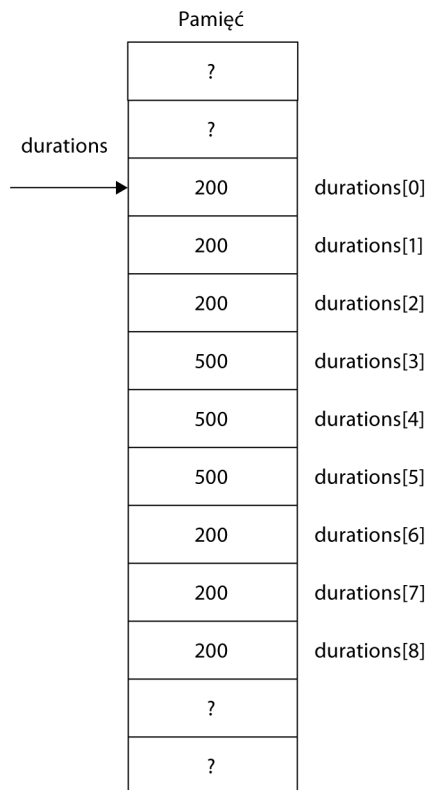
Powstał zgrabny program, który można łatwo zmodyfikować w celu dodania kolejnych wartości długości błysków. Wystarczyłoby tylko dodać je do listy znajdującej się w nawiasach klamrowych, a następnie w pętli `for` zmienić liczbę 9 na nowy rozmiar tablicy.





**Rysunek 5.1.** Monitor portu szeregowego wyświetlający dane wyjściowe szkicu 05.01.

Podczas pracy z tablicami musisz zachować pewną ostrożność. Kompilator nie powstrzyma próby uzyskania dostępu do danych znajdujących się poza obszarem tablicy. Dzieje się tak, ponieważ tablica jest tak naprawdę wskaźnikiem do adresu w pamięci, co pokazano na rysunku 5.2.



**Rysunek 5.2.** Tablice i wskaźniki

Programy przechowują dane (zarówno zwyczajne zmienne, jak i tablice) w **pamięci**. Pamięć komputera jest zorganizowana w sposób mniej elastyczny od pamięci człowieka. Pamięć Arduino można porównać do zbioru szuflad. Podczas definiowania dziewięcioelementowej tablicy rezerwowana jest przestrzeń kolejnych dziewięciu szuflad. Zmienna tablicowa jest wskaźnikiem pierwszej szuflady, zwanej pierwszym **elementem** tablicy.

Wróćmy do zagadnienia uzyskania dostępu do obszaru pamięci znajdującego się poza obszarem tablicy. Gdybyś zdecydował się uzyskać dostęp do elementu `duration[10]`, zostałaaby Ci zwrócona jakaś wartość typu `int`. Ta wartość może być jednakże dosłownie wszystkim. Uzyskanie dostępu do obszaru pamięci znajdującego się poza tablicą nie jest niczym groźnym. Uzyskasz po prostu dostęp do jakiejś przypadkowej wartości niebędącej elementem tablicy. Może to spowodować nieprawidłowe wyniki działania Twojego programu.

O wiele gorsze skutki może spowodować próba zmiany wartości znajdującej się poza obszarem tablicy. Na przykład gdybyś zamieścił w swoim programie poniższy zapis, mógłbyś poważnie zakłócić działanie programu.

```
durations[10] = 0;
```

Szuflada, w której chcemy dokonać zapisu, może już zawierać jakąś inną zmienną. A więc powinieneś bardzo uważać, żeby nie dokonać zapisu poza granicami tablicy. Jeżeli Twój szkic działa w dziwny sposób, powinieneś sprawdzić, czy nie występuje w nim problem tego typu.

## Zastosowanie tablic do alfabetu Morse'a i sygnału SOS

W szkicu 05.02. zaprezentowano zastosowanie tablic w celu wygenerowania sygnału SOS.

```
//szkic 05.02.
int ledPin = 13;

int durations[] = {200, 200, 200, 500, 500, 500, 200, 200, 200};

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  for (int i = 0; i < 9; i++)
  {
    flash(durations[i]);
  }
  delay(1000);
}

void flash(int duration)
{
  digitalWrite(ledPin, HIGH);
  delay(duration);
  digitalWrite(ledPin, LOW);
  delay(duration);
}
```

Oczywistą zaletą tej techniki jest to, że treść wiadomości można zmienić bardzo łatwo. Wystarczy po prostu zmodyfikować tablicę `durat ions`. W szkicu *05.05.* zastosujemy tablice w sposób bardziej zaawansowany — stworzymy generator alfabetu Morse’a ogólnego przeznaczenia.

## Tablice łańcuchów

W programowaniu łańcuch nie ma nic wspólnego z długim cienkim przedmiotem składającym się z ogniów. Łańcuch jest sekwencją znaków. Dzięki łańcuchom Arduino może obsługiwać teksty. Szkic *05.03.* będzie cyklicznie, co sekundę, przysyłał wiadomość tekstową o treści „Witaj” do monitora portu szeregowego:

```
// szkic 05.03.
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Witaj");
  delay(1000);
}
```

## Literały łańcuchowe

Literały łańcuchowe są zapisywane w podwójnych cudzysłowach. Literały to łańcuchy, które są stałe, w przeciwieństwie do np. zmiennej typu `int`, którą chcemy później modyfikować.

Tak jak zapewne się tego spodziewasz, łańcuchy można umieszczać w zmiennych. Istnieje również zaawansowana biblioteka obsługująca łańcuchy. Na razie będziemy jednakże używać standardowych łańcuchów języka C — takich jak ten, który znajduje się w szkicu *05.03.*

W języku C literał łańcuchowy jest tak naprawdę tablicą elementów typu `char`. Element typu `char` jest trochę podobny do elementu typu `int` — jest to również cyfra — jednakże mieści się ona w zakresie od 0 do 127 i reprezentuje jeden znak. Tym znakiem może być litera alfabetu, cyfra, znak interpunkcyjny lub znak specjalny, taki jak np. znak tabulacji lub znak przesuwu o wiersz. Kody numeryczne zawarte w łańcuchach są oparte na standardzie o nazwie ASCII. Najczęściej stosowane kody ASCII przedstawiono w tabeli 5.1.

**Tabela 5.1.** Najczęściej stosowane kody ASCII

Znak	Kod ASCII (dziesiętny)
<i>a - z</i>	97 - 112
<i>A - Z</i>	65 - 90
<i>0 - 9</i>	48 - 57
<i>spacja</i>	32

Litera łańcuchowy "Witaj" jest tak naprawdę tablicą znaków, co pokazano na rysunku 5.3.

Pamięć	
W	(87)
i	(105)
t	(116)
a	(97)
j	(106)
\0	(0)

Rysunek 5.3. Litera łańcucha o treści „Witaj”

Zauważ, że litera kończy się specjalnym znakiem null (\0). Znak ten jest stosowany do oznaczania końca łańcucha.

## Zmienne łańcuchowe

Zmienne łańcuchowe są bardzo podobne do zmiennych tablicowych. Różnicą jest to, że ich wartość początkową można zdefiniować w bardzo prosty sposób:

```
char name[] = "Witaj";
```

Zapis ten definiuje tablicę znaków i inicjalizuje ją łańcuchem znaków o treści „Witaj”. Automatycznie zostanie dodana wartość zerowa (0 w kodzie ASCII) oznaczająca koniec łańcucha.

Wcześniejszy przykład był co prawda zgodny z posiadanymi przez Ciebie wiadomościami dotyczącymi łańcuchów, jednakże częściej spotykany jest następujący zapis:

```
char *name = "Witaj";
```

Jest to zapis równoznaczny z zapisem omówionym wcześniej. Znak \* symbolizuje, że mamy do czynienia ze wskaźnikiem. Wskaźnik name wskazuje na pierwszy element typu char tablicy elementów typu char. Jest to miejsce w pamięci, gdzie zapisana jest litera W.

Możesz zmodyfikować szkic 05.03. tak, aby zastosować w nim zarówno zmienne, jak i stałe łańcuchowe:

```
// szkic 05.04.
char message[] = "Witaj";

void setup()
{
  Serial.begin(9600);
}

void loop()
```

```
{
  Serial.println(message);
  delay(1000);
}
```

## Tłumacz alfabetu Morse'a

Zastosujmy naszą wiedzę dotyczącą tablic i łańcuchów w celu napisania bardziej złożonego szkicu, który będzie odbierał komunikaty przesłane za pośrednictwem monitora portu szeregowego, a następnie przekazywał je w formie alfabetu Morse'a przy użyciu wbudowanej diody LED.

Tabela 5.2. zawiera litery i cyfry stosowane w alfabecie Morse'a.

**Tabela 5.2.** Litery i cyfry w alfabecie Morse'a

A	.-	N	-.	0	-----
B	...-	O	---	1	....-
C	-.-.	P	.-.	2	..---
D	-..	Q	---.	3	...--
E	.	R	-..	4	....-
F	..-.	S	...	5	.....
G	-.	T	-	6	-....
H	....	U	..-	7	--...
I	..	V	...-	8	---..
J	....-	W	.-.	9	----.
K	-.-	X	-.-		
L	.-..	Y	-.--		
M	--	Z	---.		

Jedną z zasad tego kodu mówi, że kreska powinna być trzy razy dłuższa od kropki. Odstęp pomiędzy każdą kreską lub kropką jest równy długości kropki. Przerwa pomiędzy dwoma literami trwa tyle samo czasu co kreska. Odstęp pomiędzy dwoma słowami powinien trwać tyle samo co siedem kropek.

W tym projekcie nie będziemy dbali o interpunkcję. Samodzielne dodanie do szkicu obsługi interpunkcji byłoby jednakże bardzo ciekawym ćwiczeniem. Pełną listę znaków stosowanych w alfabecie Morse'a znajdziesz pod adresem [http://pl.wikipedia.org/wiki/Kod\\_Morse'a](http://pl.wikipedia.org/wiki/Kod_Morse'a).

## Dane

Nasz szkic będziemy budować krok po kroku. Zaczniemy od struktury danych, którą będziesz stosować do reprezentacji kodów.

Warto zrozumieć, że każdy problem natury programistycznej ma więcej niż jedno rozwiązanie. Różni programiści tworzą różne rozwiązania tych samych problemów. Nie warto jest więc myśleć, że „nigdy bym na to nie wpadł”. Prawdopodobnie wpadłbyś na coś innego, a być może lepszego. Każdy myśli według innych schematów, a zaprezentowane przeze mnie rozwiązania po prostu wpadły mi do głowy jako pierwsze.

Reprezentacja danych sprowadza się do przełożenia na język C zawartości tabeli 5.2. Utworzymy dwa oddzielne łańcuchy — jeden na litery, a drugi na liczby. Struktura danych obsługująca litery ma następującą postać:

```
char* letters[] = {
    ".-", "-...", "-.-.", "-..", ".", // A-I
    "...", "-.-.", "...", ".:",
    "-.-.", "-.-.", "-.-.", "-.-.", "-.", // J-R
    "-.-.", "-.-.", "-.-.", "-.-.",
    "...", "-.", "-.-.", "-.-.", "-.-.", // S-Z
    "-.-.", "-.-.", "-.-."
};
```

Stworzyłeś tablicę literałów łańcuchowych. A przypominam, że literał łańcuchowy jest tak naprawdę tablicą elementów typu char, a więc tak naprawdę otrzymałeś tablicę tablic — coś, co jest w pełni zgodne z zasadami języka C, a ponadto jest bardzo przydatne.

Aby odnaleźć odpowiednik litery A w alfabecie Morse’a, musisz uzyskać dostęp do `letter[0]`. W wyniku takiej operacji zostałby zwrócony łańcuch `.-`. Metoda ta nie jest efektywna, ponieważ wykorzystujesz cały bajt (8 bitów) pamięci do zapisania kreski lub kropki, które mogłyby być zapisywane przy użyciu pojedynczych bitów. Możesz to jednakże usprawiedliwić faktem, że zajmujesz tylko 90 bajtów, a dostępnych masz około 2000. Ponadto zaprezentowana technika sprawia, że kod jest łatwy do zrozumienia, co jest również ważne.

Wykonajmy podobną tablicę dla liczb:

```
char* numbers[] = {
    "-----", ".-----", "-.-----", "-...--",
    "-....-", ".-----", "-.....", "-.....", "----..", "-----"};
```

## Zmienne globalne i funkcja setup

Musisz zdefiniować parę zmiennych globalnych. Jedna z nich będzie określała czas trwania kropki w alfabecie Morse’a, a druga będzie definiowała złącze, do którego podłączona będzie dioda LED:

```
int dotDelay = 200;
int ledPin = 13;
```

Funkcja `setup` jest dosyć prosta. Musisz tylko zaprogramować złącze tak, żeby działało jako wyjście, a także uruchomić port szeregowy:

```
void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(9600);
}
```

## Funkcja loop

Teraz zaczniesz prawdziwą pracę związaną z przetwarzaniem danych. Funkcja loop ma następujący algorytm:

- Jeżeli istnieje znak do wczytania, to wczytaj go za pośrednictwem interfejsu USB:
  - Jeżeli jest to litera, wyświetl ją przy użyciu tablicy liter.
  - Jeżeli jest to liczba, wyświetl ją przy użyciu tablicy liczb.
  - Jeżeli jest to spacja, odczekaj czterokrotność czasu trwania kropki.

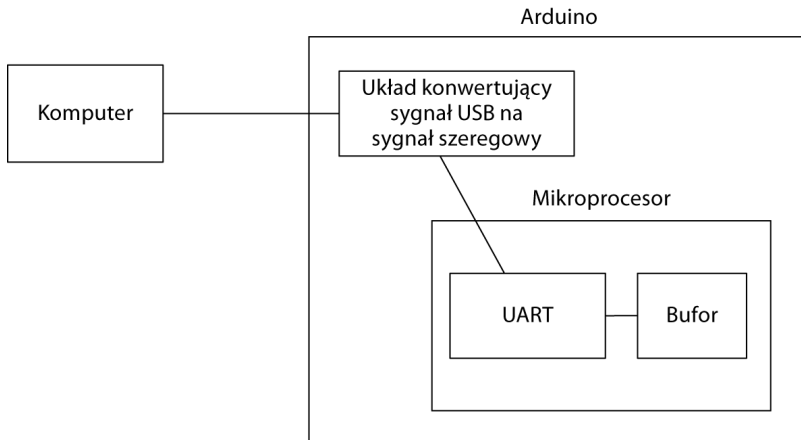
To tyle. Nie powinieneś wybiegać myślami za bardzo do przodu. Algorytm określa to, co chcesz zrobić, lub to, jakie masz **intencje**. Taki styl programowania nosi nazwę **programowania intencyjnego**.

Jeżeli przeniesiesz ten algorytm na język C, to otrzymasz następujący kod:

```
void loop()
{
    char ch;
    if (Serial.available() > 0)
    {
        ch = Serial.read();
        if (ch >= 'a' && ch <= 'z')
        {
            flashSequence(letters[ch - 'a']);
        }
        else if (ch >= 'A' && ch <= 'Z')
        {
            flashSequence(letters[ch - 'A']);
        }
        else if (ch >= '0' && ch <= '9')
        {
            flashSequence(numbers[ch - '0']);
        }
        else if (ch == ' ')
        {
            delay(dotDelay * 4); // odstęp pomiędzy słowami
        }
    }
}
```

Kilka rzeczy ujętych w tym kodzie wymaga wyjaśnienia. Pierwszą rzeczą jest zapis `Serial.available()`. Aby go zrozumieć, musisz posiadać pewną wiedzę dotyczącą komunikacji pomiędzy Arduino a komputerem poprzez złącze USB. Schemat tej komunikacji został przedstawiony na rysunku 5.4.

Dane z monitora portu szeregowego są przesyłane z komputera na płytke Arduino. Następnie sygnał zgodny ze standardem i protokołem USB jest przetwarzany na sygnał akceptowany przez mikrokontroler znajdujący się na płytce Arduino. Później specjalny układ dokonuje konwersji sygnału. Następnie dane są odbierane przez UART (*Universal Asynchronous Receiver/Transmitter* — z ang. uniwersalny asynchroniczny nadajnik i odbiornik). Jest to element mikrokontrolera, który umieszcza otrzymane dane w buforze. Bufor jest specjalnym miejscem w pamięci (128 bajtów), gdzie przechowywane są dane, które są usuwane zaraz po ich odczytaniu.



**Rysunek 5.4.** Komunikacja komputera z Arduino

Komunikacja przebiega niezależnie od wykonywanego szkicu. Dane będą dostarczane do bufora, gdzie będą czekać na odczyt, nawet podczas wykonywania operacji migania diodami LED. Pracę bufora możesz sobie wyobrazić jako działanie skrzynki odbiorczej poczty elektronicznej.

Nadejście „nowych wiadomości” sprawdzasz za pomocą funkcji `Serial.available()`. Funkcja ta zwraca liczbę bajtów danych czekających na odczyt w buforze. Jeżeli bufor jest pusty, funkcja zwraca wartość 0. Dlatego zastosowaliśmy instrukcję `if`, która sprawdza, czy do odczytu jest więcej niż 0 bajtów danych. Jeżeli w buforze znajdują się jakieś dane do odczytu, to kolejny dostępny element typu `char` jest odczytywany przy użyciu funkcji `Serial.read()`. Funkcja ta zostaje przypisana do zmiennej lokalnej `ch`.

Następnie powinniśmy umieścić kolejną instrukcję `if`, która sprawdzi, jaki znak będziesz wyświetlać za pomocą błysków diody LED:

```

if (ch >= 'a' && ch <= 'z')
{
    flashSequence(letters[ch - 'a']);
}

```

Na początku używanie operatorów `<=` i `>=` do porównywania znaków może wydawać się czymś dziwnym. Operatory te możemy jednakże stosować w ten sposób. Każdy znak jest reprezentowany liczbą (kodem ASCII). Jeżeli kod opisuje znaki od `a` do `z` (mieści się w przedziale od 97 do 122), to wiesz, że z komputera została przekazana mała litera. Następnie możesz wywołać nienapisaną jeszcze funkcję `flash.Sequence`. Do tej funkcji przekażemy łańcuch składający się z kropek i kresek. Na przykład w celu wyświetlenia za pomocą błysków diody LED litery `a` przekażemy do tej funkcji w roli argumentu ciąg `.-`.

Przenosisz do tej funkcji całkowitą odpowiedzialność za wykonywanie błysków diodą. Nie próbuj przeprowadzać tej operacji wewnątrz funkcji `loop`. W ten sposób kod będzie bardziej przejrzysty.



Poniżej znajduje się kod określający ciąg kropek i kresek, które następnie będziesz musiał przekazać funkcji `flashSequence`:

```
letters[ch - 'a']
```

Również ten zapis może wydawać się dziwny. Funkcja wydaje się odejmować od siebie znaki. Zapis ten jest jednakże poprawny. W naszym programie odejmujemy wartości będące kodami ASCII.

Pamiętaj o tym, że kody liter przechowujesz w tablicy, a więc pierwszy element tablicy przechowuje ciąg kropek i kresek symbolizujących literę *A*. Drugi element tablicy przechowuje ciąg symbolizujący literę *B* itd. Musisz odnaleźć pozycję w tablicy, której odpowiada litera pobrana z bufora. Pozycja małych liter będzie taka sama jak pozycja liter wielkich. Wystarczy przeprowadzić proste działanie matematyczne polegające na odjęciu wartości kodu symbolizującego literę *a*. Działanie  $a-a$  to tak naprawdę operacja  $97-97 = 0$ . Podobnie  $c-a$  to tak naprawdę  $99-97 = 2$ . Jeżeli w przytoczonym powyżej kodzie zmienna `ch` będzie przechowywała literę *c*, to działanie umieszczone w nawiasie da w wyniku wartość 2, a z tablicy zostanie pobrany element o numerze 2, czyli ciąg `.-.-`.

Podrozdział, który właśnie przeczytałeś, opisuje obsługę małych liter. Musisz także obsłużyć wielkie litery i cyfry. Ich obsługa będzie przebiegała w podobny sposób.

## Funkcja `flashSequence`

Wcześniej zakładaliśmy istnienie funkcji `flashSequence` i stosowaliśmy ją w kodzie programu. Teraz musimy ją napisać. Planowaliśmy, że funkcji tej będziemy przekazywać łańcuch składający się z serii kropek i kresek, a funkcja ta będzie wykonywała odpowiednią liczbę błysków w odpowiednim czasie.

Obmyślając algorytm tej funkcji, możesz podzielić go na dwa etapy:

- Dla każdego elementu łańcucha kresek i kropek (np. `.-.-`)
- wykonuj odpowiednio długi błysk.

Stosując zasady programowania intencyjnego, postarajmy się, aby funkcja była jak najprostsza.

Kody Morse'a różnych liter mają różną długość. Będziesz więc musiał zastosować pętlę przeglądającą łańcuch aż do napotkania znaku `\0`. Będzie Ci również potrzebna zmienna o nazwie `i`, o początkowej wartości 0. Wartość tej zmiennej będzie zwiększana podczas przeglądania kolejnych kropek i kresek:

```
void flashSequence(char* sequence)
{
    int i = 0;
    while (sequence[i] != '\0')
    {
        flashDotOrDash(sequence[i]);
        i++;
    }
    delay(dotDelay * 3); // odstęp pomiędzy literami
}
```

Odpowiedzialność za włączanie i wyłączenie diody LED zostaje przeniesiona na nową funkcję o nazwie `flashDotOrDash`. Gdy program wygeneruje odpowiednie błyski dla wszystkich kropek i kresek, następuje pauza o długości trzech kropek. Zauważ, jak przydatne w tym miejscu okazało się zastosowanie komentarza.

## Funkcja `flashDotOrDash`

Ostatnia brakująca funkcja będzie bezpośrednio włączała i wyłączała diodę LED. Funkcja ta przyjmuje pojedynczy argument, który będzie kropką (.) lub kreską (-).

Funkcja włącza diodę LED i oczekuje określony okres czasu, jeżeli do funkcji została przekazana kropka. Jeżeli do funkcji została przekazana kreska, funkcja odczekuje trzy razy dłuższy okres czasu. Po tym czasie następuje wyłączenie diody LED. Na koniec funkcja musi odczekać czas równy czasowi „wyswietlania” kropki — w ten sposób zaznaczany jest odstęp pomiędzy błyskami.

```
void flashDotOrDash(char dotOrDash)
{
    digitalWrite(ledPin, HIGH);
    if (dotOrDash == '.')
    {
        delay(dotDelay);
    }
    else // jeżeli nie jest kropką, to musi być kreską
    {
        delay(dotDelay * 3);
    }
    digitalWrite(ledPin, LOW);
    delay(dotDelay); // odstęp pomiędzy błyskami
}
```

## Składanie całości programu

Cały program zawierający wszystkie funkcje znajduje się w szkicu *05.05*. Załaduj go na płytkę Arduino i uruchom. Pamiętaj, że aby korzystać z tego szkicu, musisz otworzyć monitor portu szeregowego, wpisać jakiś tekst, a następnie kliknąć ikonę *Send*. Wpisany przez Ciebie tekst powinien zostać przekształcony na alfabet Morse’a przy użyciu wbudowanej w Arduino diody LED.

```
// szkic 05.05.
int ledPin = 13;
int dotDelay = 200;

char* letters[] = {
    ".-", "-...", "-.-.", "-..", ". .", ".-.-", "--", "...", ". .", // A-I
    "-.-", "-.-", "-.-.", "-.", "-.", "-.-", "-.-.", "-.-.", // J-R
    "...", "-.", "-.-", "-.-.", "-.-", "-.-.", "-.-.", "-.-." // S-Z
};

char* numbers[] = {
    "-----", ".-----", "-.-----", "-.-.-.-", "-.-.-.-", ".-----", "-.-.-.-", "-.-.-.-", "-.-.-.-"};
```

```

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  char ch;
  if (Serial.available() > 0)
  {
    ch = Serial.read();
    if (ch >= 'a' && ch <= 'z')
    {
      flashSequence(letters[ch - 'a']);
    }
    else if (ch >= 'A' && ch <= 'Z')
    {
      flashSequence(letters[ch - 'A']);
    }
    else if (ch >= '0' && ch <= '9')
    {
      flashSequence(numbers[ch - '0']);
    }
    else if (ch == ' ')
    {
      delay(dotDelay * 4); // odstęp pomiędzy błyskami
    }
  }
}

void flashSequence(char* sequence)
{
  int i = 0;
  while (sequence[i] != NULL)
  {
    flashDotOrDash(sequence[i]);
    i++;
  }
  delay(dotDelay * 3); // odstęp pomiędzy literami
}

void flashDotOrDash(char dotOrDash)
{
  digitalWrite(ledPin, HIGH);
  if (dotOrDash == '.')
  {
    delay(dotDelay);
  }
  else // jeżeli nie jest kropką, to musi być kreską
  {
    delay(dotDelay * 3);
  }
  digitalWrite(ledPin, LOW);
  delay(dotDelay); // odstęp pomiędzy błyskami
}

```

W szkicu umieszczono funkcję `loop`, która jest wywoływana automatycznie. Funkcja ta wywołuje napisaną przez Ciebie funkcję `flashSequence`, która to z kolei wywołuje napisaną przez Ciebie funkcję `flashDotOrDash`, a ta wywołuje funkcje `digitalWrite` i `delay`, które to są funkcjami „wbudowanymi” w Arduino.

Tak powinny wyglądać Twoje szkice. Podział szkicu na funkcje pozwala na łatwiejszą implementację kodu. Ułatwia on również powrót do pracy nad szkicem po dłuższej przerwie.

---

## Podsumowanie

W tym rozdziale poznałeś podstawowe wiadomości dotyczące łańcuchów i tablic. Napisałeś dość złożony szkic tłumaczący wprowadzany tekst na alfabet Morse’a. Mam nadzieję, że dzięki temu szkicowi zrozumiałeś, jak ważny jest podział programu na funkcje.

W kolejnym rozdziale opiszemy zagadnienia związane z wejściami i wyjściami. Omówimy wprowadzanie do Arduino, a także wyprowadzanie z niego, sygnałów analogowych i cyfrowych.

# Skorowidz

## A

adres MAC, 124  
akapit, 123  
alfabet Morse'a, 67, 71, 73, 107  
algorytm, 67, 75  
antena, 85  
Arduino  
  aplikacja, 33  
  dokumentacja, 19  
  historia, 19  
  uruchamianie, 27

## B

Banz Massimo, 19  
biblioteka, 91  
  Bounce, 91, 92  
  Ethernet, 126  
  funkcji matematycznych, 99  
  nieoficjalna, 108  
  obsługująca wyświetlacze  
    LCD, 116, 134  
  pgmspace, 108  
  Processing, 97  
  PROGMEM, 108  
  tworzenie, 134  
  Wiring, 97  
bit, 99  
Boole George, 60  
bufor, 75

## C

camelCase, 42  
CPU, 18  
Cuartielles David, 19

## D

dane  
  kompresja, 112, 113  
  struktura, 67  
  typ, Patrz: zmienna typ  
  zapis  
    do pamięci EEPROM,  
      109, 110, 111, 112  
    do pamięci flash, 108  
    unia, 110  
  zapisywanie, 107  
debugowanie, 56  
dioda LED, Patrz: LED  
dyrektywa  
  #define, 52, 56  
  #include, 92  
  PROGMEM, 107, 108, 112

## E

EEPROM, 15, 18, 109  
  wymazywanie, 112  
  zapis danych, 109, 110,  
    111, 112  
Ethernet, 121

## F

formularz, 126  
funkcja, 53, 58  
  abs, 100  
  argument, 36, 58  
  bitRead, 101  
  bitWrite, 102  
  client.connected, 126  
  client.stop, 126  
  constrain, 100  
  cos, 100  
  definiowanie, 40  
  delay, 39, 51  
  digitalWrite, 36, 39, 88  
  EEPROM.write, 109  
  highByte, 110  
  interrupts, 104  
  log, 100  
  loop, 38, 39, 40, 41, 75  
  lowByte, 110  
  map, 100  
  matematyczna, 47, 99  
  max, 100  
  millis, 126  
  min, 100  
  noTone, 103  
  pageNameIs, 130  
  parametr, 127  
  parametry, 54, 55  
  pgm\_read\_word, 108  
  pinMode, 40, 83

## funkcja

pow, 100  
 random, 98  
 readHeader, 130  
 Serial.available, 75, 76  
 Serial.println, 45  
 server.available, 126  
 setup, 38, 39, 40, 44, 74  
 setValuesFromParams, 130  
 shiftOut, 103  
 sin, 100  
 sqrt, 100  
 tan, 100  
 tone, 102  
 trygonometryczna, 100  
 typu int, 58  
 typu void, 58  
 valueOfParam, 130  
 wartość zwracana, 58  
 wejścia, 102  
 writeHTMLforPin, 130  
 wyjścia, 102  
 wywołanie, 36  
 wywoływanie, 40

**G**

## generator

alfabetu Morse'a, 71  
 drgań, 19  
 liczb losowych, 97, 99  
 osadzanie, 99  
 sprzętowy, 99

**H**

hermetyzacja, 56, 133  
 HTML, 122  
 znacznik, Patrz: znacznik

**I**

interfejs, 15  
 USB, 19, 45, 97  
 interferencja  
 elektromagnetyczna, 87

**J**

jednostka centralna, Patrz: CPU  
 język  
 C, 35  
 funkcja wbudowana, 36  
 nazwa, 36  
 składnia, 36, 42  
 C++, 93, 133  
 hipertekstowego  
 znakowania informacji,  
 Patrz: HTML  
 Java, 97

**K**

klasa, 133  
 LiquidCrystal, 134  
 kod, Patrz też: szkic  
 ASCII, 71, 76  
 blok, 40  
 HTML, Patrz: HTML  
 testowanie, 44  
 wcięcia, 62  
 zapis, 62  
 komentarz, 64  
 komunikat błędu, 37  
 konstruktor, 135, 136  
 kontroler Bluetooth, 23

**L**

LED, 78  
 liczba  
 całkowita, 42  
 losowa, 97  
 mapowanie, 100  
 ograniczenie, 100  
 pseudolosowa, 98  
 w systemie dwójkowym,  
 Patrz: bit  
 zmiennoprzecinkowa, 59  
 licznik, 50  
 literał łańcuchowy, 71

**M**

mechanizm obiektowy, 133  
 metoda, 133, 135  
 Flasher, 135  
 prywatna, 133  
 publiczna, 133

## mikrokontroler

ATmega1280, 22  
 ATmega168, 21  
 ATmega328, 16, 18, 21  
 bufor, Patrz: bufor  
 PIC, 25  
 wymiana, 22  
 modem USB, 29  
 modulacja czasu trwania  
 impulsu, Patrz: PWM  
 moduł wyświetlacza LCD,  
 Patrz: wyświetlacz LCD  
 monitor portu szeregowego,  
 44, 83, 85  
 multimetr, 81

**N**

nagłówek, 123, 130  
 numer IP, 124

**O**

obiekt bouncer, 93  
 operacja  
 dodawania, 47  
 dzielenia, 47  
 iloczynu logicznego, 60  
 logiczna, 60  
 matematyczna, 97, 99  
 kolejność, 47  
 mnożenia, 47  
 na bitach, 97  
 odejmowania, 47  
 porównywania, 60  
 sumy logicznej, 60  
 operator porównywania, 48  
 opóźnienie, 91  
 oprogramowania  
 instalowanie, 28

**P**

pamięć, 70  
 flash, 18, 107  
 zapis danych, 108  
 o dostępie bezpośrednim,  
 Patrz: RAM  
 operacyjna, 19  
 robocza, 18  
 stała programowalna,  
 Patrz: EEPROM

- pętla  
 for, 49  
 argumenty, 50  
 while, 51
- pierwiastek, 100
- pin, Patrz: złącze
- plik  
 .cpp, 134, 136  
 .h, 134  
 implementacji, 134, 136  
 LiquidCrystal.h, 134  
 nagłówkowy, 134, 136
- płyta  
 Arduino, 17, 24, 25  
 nieoficjalna, 25  
 zasilanie, Patrz: zasilanie  
 złącze analogowe, Patrz:  
 złącze analogowe  
 złącze cyfrowe, Patrz:  
 złącze cyfrowe  
 Arduino Bluetooth, 23  
 Arduino Lilypad, 24  
 Arduino Mega, 22  
 Arduino Nano, 22  
 Arduino Uno, 21, 22, 29  
 Chipkit, 25  
 Diecimila, 21  
 Duemilanove, 21  
 Femtoarduino, 25  
 Freeduino, 25  
 Roboduino, 25  
 rozwojowa, 16  
 Ruggeduino, 25  
 Seeeduino, 25  
 stykowa, 20, 22  
 Ethernet, Patrz: Ethernet  
 Host USB, Patrz: Host  
 USB  
 Motor, Patrz: Motor  
 Teensy, 25  
 wyświetlacza LCD, Patrz:  
 wyświetlacz LCD
- polecenie, 47  
 if, 47, 48, 49  
 warunek, Patrz: warunek  
 include, 134  
 int, 113  
 return, 58
- port  
 COM3, 29  
 szeregowy, 19, 29  
 monitor, Patrz: monitor  
 portu szeregowego  
 program, Patrz: szkic  
 programator zewnętrzny, 24  
 programowanie intencyjne, 75  
 protokół  
 DHCP, 124  
 HTTP, 122  
 przeglądarka, 126  
 przekaźnik, 20  
 przełącznik, 85  
 dotykowy, 88  
 typu klik, 88  
 przerwanie, 103  
 CHANGE, 104  
 FALLING, 104  
 RISING, 104  
 przypisanie, 45, 49  
 Pulse Width Modulation,  
 Patrz: PWM  
 PWM, 93, 95
- ## R
- RAM, 15, 18, 107  
 regulator  
 napięcia, 17  
 rezystor  
 podwyższający, 85, 89  
 wewnętrzny, 88  
 rzutowanie, 113
- ## S
- serwer  
 sieci Web, 123  
 wyszukiwarki Google, 123  
 shield, Patrz: płyta stykowa  
 silnik elektryczny, 18, 20  
 słowo kluczowe  
 int, 58  
 PROGMEM, 108  
 static, 57  
 void, 39, 40  
 stała, 56, 109  
 standard ASCII, Patrz: kod  
 ASCII  
 stuki, 89  
 redukcja, 91  
 sygnał przzerwania, 104
- system  
 dwójkowy, 99  
 heksadecymalny, Patrz:  
 system szesnastkowy  
 szesnastkowy, 101  
 szkic, 18, Patrz też: kod  
 Blink, 27  
 kompilacja, 32, 37, 39  
 ładowanie, 28, 32, 37  
 wbudowany, 33  
 wykonywanie, 32  
 szkieletownik, 33
- ## T
- tablica, 67  
 element, 68, 70  
 typu char, 71  
 indeks, 67  
 łańcuchów, 71  
 tablic, 74  
 znaków, 108  
 temperatura, 45
- ## U
- UART, 75  
 układ  
 ENC28J60, 122  
 HD44780, 115  
 regulatora napięcia, Patrz:  
 regulator napięcia  
 rejestru przesuwne, 103  
 unia, 110  
 Universal Asynchronous  
 Receiver/Transmitter,  
 Patrz: UART  
 urządzenie sieciowe, 124
- ## W
- wartość logiczna, 60  
 wejście, Patrz: złącze:wejściowe  
 Wirth Niklaus, 67  
 wyjście, Patrz: złącze:wyjściowe  
 wyświetlacz LCD, 115, 116
- ## Z
- zasilanie, 122  
 złącze, 17  
 zegar, 103

- złącze
    - analogowe, 18
    - cyfrowe, 18
    - D2, 104
    - D3, 104
    - GND, 85
    - sterowanie, 130
    - zasilające, 17
    - USB, 21, 23, 24, 75
    - wejściowe, 40, 85
      - analogowe, 81
      - cyfrowe, 81, 84
      - pływające, 85, 99
    - włączanie, 126
    - wyjściowe, 40
      - 13, 35
      - analogowe, 81, 93
      - cyfrowe, 81, 95
    - wyłączanie, 126
  - zmienna, 42, 45
    - boolean, 60, 61
    - byte, 61
    - char, 61, 71
    - deklarowanie, 46
    - double, 61
    - float, 59, 61, 112
    - globalna, 55, 56, 74
    - int, 42, 46, 59, 61, 100, 112
    - lokalna, 56
      - inicjalizowanie, 56, 57
      - long, 61
      - łańcuchowa, 72, 111
      - nazwa, 42
      - składowa, 133
      - tablicowa, 68, 70
      - typ, 61
        - konwersja, 113
      - unsigned int, 61
      - unsigned long, 61
      - wartość początkowa, 46
  - znacznik
    - body, 123, 126
    - h1, 123, 126
    - html, 123, 126
    - otwierający, 123
    - p, 123, 126
    - zamykający, 123
  - znak
    - !=, 48
    - &, 130
    - &&, 60
    - \*/, 64
    - /\*, 64
    - //, 64
    - ?, 130
    - ||, 60
    - <, 48, 123
    - <=, 48
    - =, 49, 52
    - ==, 48, 49
    - >, 48, 123
    - >=, 48
    - biały, 63
    - cudzysłowu podwójnego, 71
    - gwiazdki, 47
    - końca łańcucha, 72
    - kreski ukośnej, 47
    - nawiasu, 40, 47
    - nawiasu klamrowego, 40, 63, 68
    - nawiasu kwadratowego, 68
    - nowego wiersza, 63
    - null, 72
    - przecinka, 36, 50
    - równości, 45
    - separatora dziesiętnego, 59
    - spacji, 62, 63
    - średnika, 36, 40, 50, 52
    - tabulacji, 62, 63
    - większości, 48
- Ż**
- żądanie, 127



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

# Lektura obowiązkowa każdego adepta platformy Arduino!

Jeśli nie po drodze Ci z lutownicą i trawieniem płytek, a Twoją pasją jest elektronika, Arduino to platforma dla Ciebie! Dzięki niej rozwiniesz skrzydła i zrealizujesz swoje wymarzone projekty. Do rynkowego sukcesu Arduino z pewnością przyczyniły się świetna dokumentacja, obszerne źródła informacji oraz środowisko przeznaczone specjalnie do tworzenia oprogramowania. Obecnie dzięki tej platformie oraz dodatkowym akcesoriom jesteś w stanie zbudować praktycznie dowolny projekt — nieważne, czy jest to licznik Geigera, czy sterownik oświetlenia LED.

W tej książce główny nacisk został położony na aspekty związane z programowaniem Arduino. W trakcie lektury opanujesz podstawy języka C, a poza tym nauczysz się używać struktur danych oraz korzystać z analogowych i cyfrowych złączy znajdujących się na płytce Arduino. Ponadto poznasz możliwości standardowej biblioteki Arduino, sposoby zapisywania programu oraz techniki wyświetlania informacji na ekranach LCD. Na koniec dowiesz się, jak podpiąć płytkę Arduino do sieci, a następnie zaprogramować aplikację sieci Ethernet. Książka ta jest doskonałym podręcznikiem dla wszystkich pasjonatów elektroniki chcących lepiej poznać platformę Arduino.

## Dzięki tej książce:

- zaznajomisz się z konstrukcją płytki Arduino
- poznasz podstawy języka C wymagane do programowania Arduino
- odbierzesz i wyślesz dane przez interfejs wejścia-wyjścia
- rozpoczniesz przygodę z platformą Arduino

**helion.pl**  
księgarnia  
internetowa

Nr katalogowy: 17859

Księgarnia internetowa:  
<http://helion.pl>

Zamówienia telefoniczne:  
**0 801 339900**  
**0 601 339900**



**Helion**

Sprawdź najnowsze promocje:  
• <http://helion.pl/promocje>  
Książki najchętniej czytane:  
• <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
• <http://helion.pl/nowosci>

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-8707-7



cena: 29,90 zł

Informatyka w najlepszym wydaniu

9 788324 687077