Rejestrowanie danych

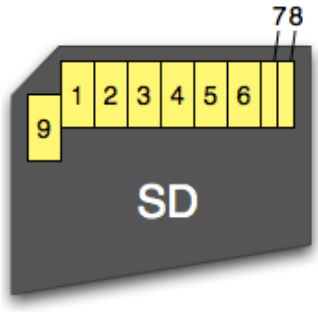
Czujniki pozwalają uzyskiwać informację o otaczającym świecie. Najczęściej nie są to pojedyncze odczyty lecz zbierane są informacje o zmianach badanych wielkości w dłuższym okresie czasu. Prowadzi to do rejestracji dużej ilości danych, które następnie mogą być wykorzystane do analizy stanu badanego obiektu lub zjawiska. Badane są również relacje pomiędzy danymi odbieranymi z czujników.

Rodzaje pamięci

Aby rejestrować dane lokalnie na Arduino, należy mieć możliwość ich trwałego zapisu i przechowywania. Zazwyczaj wykorzystuje się w tym celu jakąś zewnętrzną pamięć, na przykład w komputerze, telefonie komórkowym lub pendrivie. Pamięć zewnętrzna ma tą cenną cechę, że dane w niej zapisane pozostają zapamiętane nawet po wyłączeniu zasilania. Ta zasada dotyczy również wbudowanej w Arduino pamięci EEPROM.

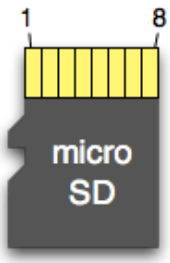
Zawartość pamięci EEPROM jest przechowywana nawet po wyłączeniu zasilania Arduino, ale jej wielkość jest bardzo ograniczona. W zależności od modelu płyty dostępnych jest od 512 do 4096 bajtów. Dlatego pamięć EEPROM nadaje się do przechowywania zmiennych i niewielkich ilości danych, które muszą być dostępne pomiędzy kolejnymi włączeniami Arduino, ale nie nadaje się do większości zastosowań obejmujących rejestrowanie danych. Stąd często korzysta się z zewnętrznych rozwiązań, takich jak karta SD (ang. *Secure Digital*, bezpieczny cyfrowy zapis) lub pamięć USB.

Użycie karty SD jest jednym z najlepszych sposobów przechowywania danych w pamięci zewnętrznej. Dostępne są karty o dużych pojemnościach, oferujących mnóstwo przestrzeni nie tylko na rejestrowane dane, ale również na innego rodzaju pliki (na przykład muzykę lub obrazy). Dla płyty Arduino jest dostępna biblioteka *SD*, dzięki której odczytywanie i zapis danych na karcie SD są proste.



Pin	SD	SPI
1	CD/DAT3	CS
2	CMD	DI
3	VSS1	VSS1
4	VDD	VDD
5	CLK	SCLK
6	VSS2	VSS2
7	DAT0	DO
8	DAT1	X
9	DAT2	X

Karta SD, a dokładniej Standard SD (ang. secure digital) to jeden ze standardów kart pamięci opracowany przez firmy Panasonic, SanDisk i Toshiba w 2000 roku. Karty SD charakteryzują się niewielkimi wymiarami (24 × 32 × 2,1 mm).



Pin	SD	SPI
1	DAT2	X
2	CD/DAT3	CS
3	CMD	DI
4	VDD	VDD
5	CLK	SCLK
6	VSS	VSS
7	DAT0	DO
8	DAT1	X

Karty SD posiadają 9 wyprowadzeń. Dodatkowo karta SD ma na brzegu przełącznik blokujący możliwość zapisu.

Karta SD operuje na napięciu 3,3V.

Przy zakupie kart mamy do wyboru rodzaj standardu oraz klasę karty. Można powiedzieć, że standard karty oznacza między innymi zakres pamięci karty

- SD (8MB – 2GB)
- SDHC (4GB – 32GB)
- SDXC (64GB – 2TB)

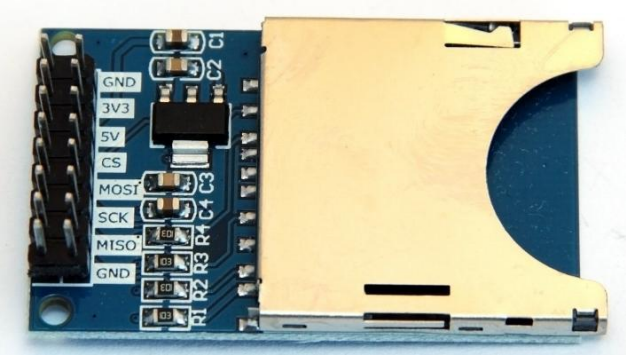
Natomiast klasa karty określa prędkość odczytu i zapisu z karty (tutaj panuje zasada „im wyższy numer tym lepsza”):

odczyt (zapis)

- klasa 2: 16 Mb/s (2 MB/s)
- klasa 4: 32 Mb/s (4 MB/s)
- klasa 6: 48 Mb/s (6 MB/s)
- klasa 10: 80 Mb/s (10 MB/s)

Przed użyciem karta SD powinna być sformatowana. Spowoduje to utworzenie systemu plików z którego Arduino może skorzystać. Arduino akceptuje format FAT16/32.

Aby w systemie mikroprocesorowym skorzystać z kart SD można w tym celu użyć modułu czytnika kart SD lub czytnika wbudowanego do karty Ethernet.



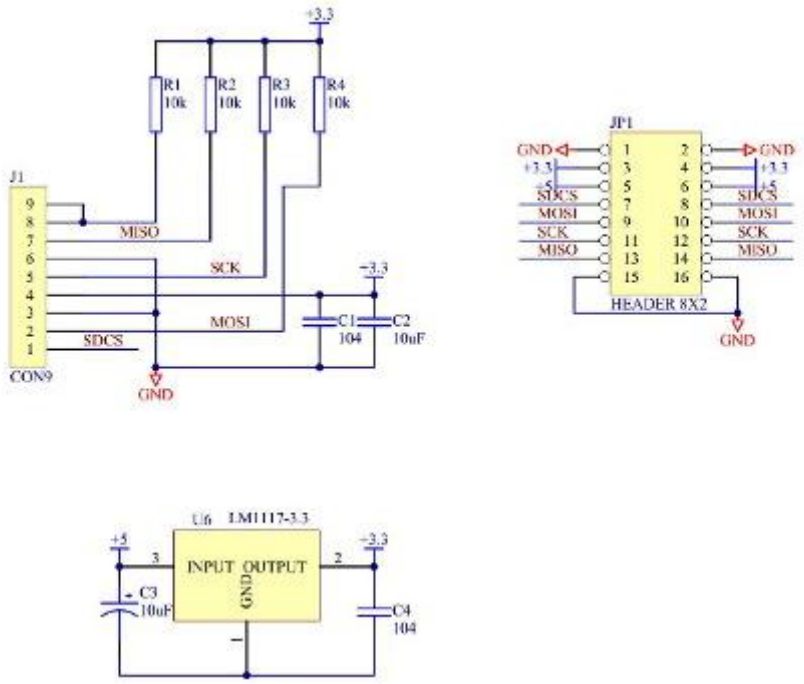
Opis modułu czytnika kart SD
Moduł ułatwiający podłączenie karty SD do płytki głównej. Zasilany napięciem 5 V lub 3,3 V. Interfejsem komunikacyjnym jest magistrala szeregowa SPI. Wyprowadzeniami są popularne złącza goldpin, umożliwiające podłączenie karty pamięci do zestawu uruchomieniowego Arduino za pomocą przewodów.

Uwaga: Karty SD standardowo pracują z napięciem 3,3 V, dlatego też w celu połączenia czytnika z Arduino należy zastosować konwerter napięć.

Specyfikacja

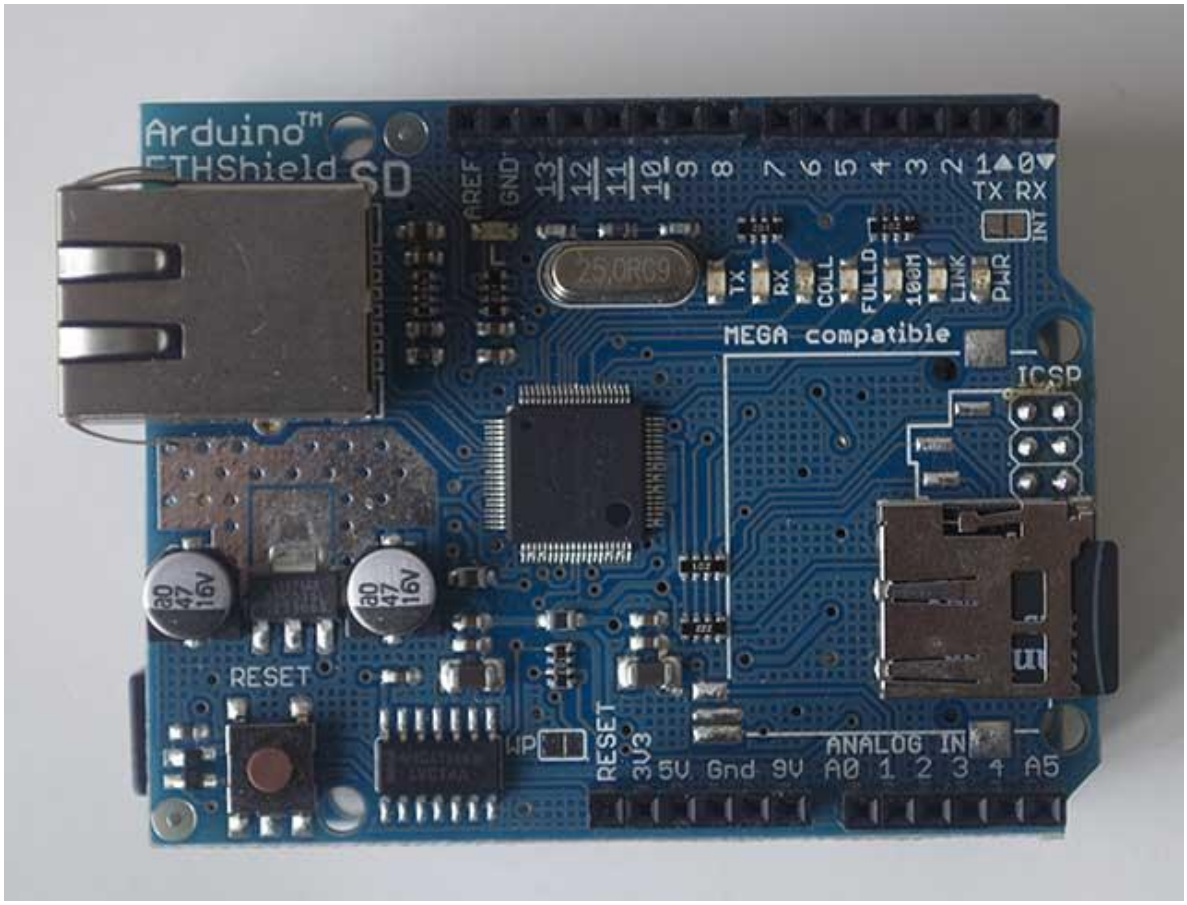
- Napięcie zasilania: 3,3 V lub 5 V
- Interfejs komunikacyjny: SPI
- Wbudowany stabilizator napięcia 3,3 V (LM1117)
- Zamontowane gniazdo kart SD z wyrzutnikiem
- Wymiary modułu: 50 x 31 mm

Na płytce znajdują się: gniazdo na karty SD z wyrzutnikiem, stabilizator napięcia oraz niezbędne do poprawnego działania układu elementy pasywne.



Schemat modułu czytnika karty SD

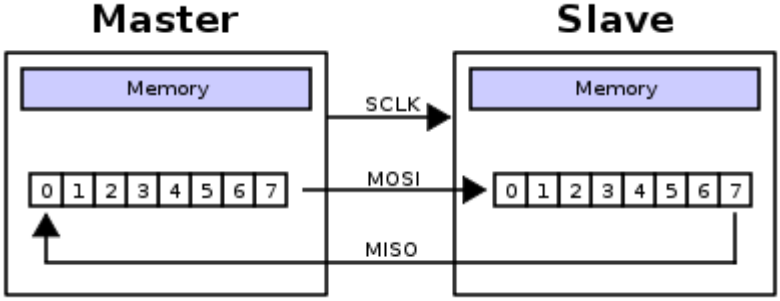
Innym rozwiązaniem jest zastosowanie nakładki Ethernet z wbudowanym czytnikiem kart SD.



Karta Ethernet z wbudowanym czytnikiem kart mikroSD

Komunikacja pomiędzy mikrokontrolerem, a czytnikiem kart odbywa się za pośrednictwem interfejsu szeregowego SPI, który wykorzystuje cyfrowe piny 11, 12 i 13 w Arduino Uno oraz 50, 51 i 52 w Arduino Mega. Dodatkowo jest wykorzystywany jeden pin 4 (SS) do wyboru układu w interfejsie SPI i wyspecyfikowany w funkcji SD.begin(). Uwaga: pin ten musi być ustawiony jako wyjście nawet gdy nie jest wykorzystywany do wyboru układu.

Transmisja w interfejsie SPI jest transmisją szeregową synchroniczną, która pozwala komunikować się na krótkie odległości mikrokontrolerowi z jednym lub kilkoma urządzeniami. Umożliwia również transmisję między dwoma mikroprocesorami.



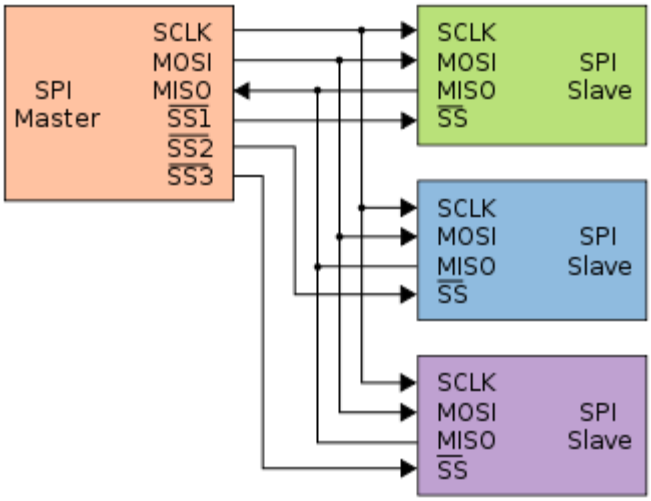
W interfejsie SPI jedno urządzenie pełni funkcję **master** (zwykle jest to mikrokontroler) i steruje pracą urządzeń typu **slave**. Trzy linie są wspólne dla wszystkich urządzeń:

- **MISO** (Master In Slave Out) - linią tą urządzenie slave przesyła dane do urządzenia master,
- **MOSI** (Master Out Slave In) - linią tą urządzenie master przesyła dane do urządzenia slave,
- **SCK** (Serial Clock) – impulsy zegarowe generowane przez urządzenie master służące do synchronizacji transmisji danych

i jedna linia wydzielona dla każdego urządzenia slave:

- **SS** (Slave Select) - linia za pomocą której urządzenie master wybiera urządzenie slave do współpracy (gdy stan na tej linii jest niski).

Mikrokontroler – układ Master, może współpracować z kilkoma układami typu Slave, ale nie jednocześnie. Wybór układu następuje przez podanie stanu niskiego na wyjście SS. Taka sytuacja ma miejsce np. w nakładce Ethernet gdzie pin cyfrowy 10 służy do wyboru sterownika karty Ethernet, zaś 4 do wyboru czytnika kart SD.



Zasady działania interfejsu SPI znajdują się na stronie https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

Opis funkcji Arduino z biblioteki SPI zawiera strona https://www.arduino.cc/en/Reference/SPI

Biblioteka SD opisana jest na stronie https://www.arduino.cc/en/Reference/SD

Karty SD i biblioteka SD

Tabela 1 zawiera przegląd najważniejszych funkcji klasy SD z biblioteki *SD*, ale dostępnych jest znacznie więcej funkcji klasy File, służących do odczytu i zapisu plików.
Tabela 2 zawiera listę najważniejszych funkcji tej klasy, natomiast pełna lista jest dostępna w dokumentacji w Internecie pod adresem *www.arduino.cc/en/Reference/SD*.
Niezależnie od tego, czy używasz nakładki SD, czy czytnika wbudowanego w nakładkę Ethernet, biblioteki *SD* używa się w taki sam sposób.

1. Funkcje klasy SD z biblioteki SD

Funkcja	Opis
begin(chipSelect)	Inicjuje bibliotekę *SD* i kartę, opcjonalnie można wskazać pin do wyboru układu
exists()	Sprawdza, czy plik lub katalog istnieje na karcie SD
mkdir("/*katalog/do/utworzenia*")	Tworzy katalog na karcie SD
rmdir("/*katalog/do/usuniecia*")	Usuwa katalog z karty SD
open("*plik/do/otwarcia*", tryb)	Otwiera plik o zadanej ścieżce
remove("*plik/do/usuniecia*")	Usuwa plik o zadanej ścieżce

Przy otwieraniu pliku można określić tryb FILE_READ (odczyt) lub FILE_WRITE (zapis), jeżeli zachodzi potrzeba ograniczenia do niego dostępu tylko do odczytu lub tylko do zapisu.

2. Funkcje klasy File z biblioteki SD

Funkcja	Opis
available()	Sprawdza, czy w pliku dostępne są bajty do odczytu
close()	Zamyka plik i sprawdza, czy wszystkie dane zostały zapisane na karcie SD
flush()	Zapisuje fizycznie dane na karcie SD. Funkcja wykorzystywana przez close()
print()	Zapisuje dane do pliku
println()	
write()	
read()	Odczytuje bajt z pliku

Podstawowe funkcje do obsługi karty

(na podstawie http://feriar-lab.pl/kurs-arduino-16-obsluga-kart-sd/)

Teraz opiszę funkcję, które musimy znać, aby być w stanie bez problemu obsługiwać kartę SD.

Na początku musimy dodać biblioteki obsługujące interfejs SPI oraz standard SD:

```
#include  <SPI.h>

#include  <SD.h>
```

Następnie tworzymy instancję do klasy *File*, która jest odpowiedzialna m.in. za operacje na plikach:

File plik;

Instancji *File* możemy nadać dowolną nazwę, ja nadałem po prostu nazwę *plik*.

Uruchamiamy obsługę karty. Można to zrobić na dwa sposoby:

1. Dajemy Arduino bezpośrednią informację, że karta jest podłączona. On wtedy nie sprawdza stanu faktycznego tylko próbuje wykonywać program
2. Arduino sprawdzi czy karta jest podłączona do Arduino i wtedy podejmie decyzję co zrobić.

W pierwszym przypadku po prostu piszemy:

```
SD.begin(pin_CS)       //pin_CS - numer pinu do którego podpięty jest sygnał CS z karty SD
```

Drugi sposób opiera się na sprawdzeniu warunku:

```
if (!SD.begin(4))                          //sprawdź czy nie ma karty na pinie ChipSelect 4
 {
   Serial.println("Nie wykryto karty");    //błąd wykrycia karty
   return;                                 //przerwij program
 }
```

Sprawdź czy nie ma karty, wstawiając operator zaprzeczenia „!”. Jeżeli karta nie zostanie wykryta to wydrukuj komunikat i przerwij wykonywanie programu przy pomocy funkcji *return*. Wtedy dalsza część programu nie będzie wykonywana. Dopiero po włożeniu karty i ponownym zresetowaniu Arduino będzie wykonywana dalsza część programu.

Teraz, gdy Arduino już wie, czy karta jest podłączona czy nie to możemy przejść do wykonywania operacji na plikach.

Możemy sprawdzić czy w pamięci karty znajduje się dany plik, w następujący sposób:

```
SD.exists("nazwa_pliku")
```

W nazwie pliku musi się znajdować również rozszerzenie. Ważne aby nazwy plików nie były dłuższe niż 8 znaków i nie zawierały specjalnych znaków.

Kolejną funkcją tworzymy plik, otwieramy oraz przyznajemy prawa zapisu i odczytu danych:

```
plik = SD.open("nazwa_pliku", FILE_WRITE);
```

plik to reprezentacja instancji, w nazwie pliku podajemy również nazwę maksymalnie 8 znakową wraz z rozszerzeniem, a przy pomocy *FILE_WRITE* przyznajemy prawa zapisu i odczytu. Możemy jeszcze użyć *FILE_READ*, ale wtedy przyznamy tylko prawa do odczytu danych z pliku. Przy operacji na plikach ważne jest, iż jednocześnie może być otwarty tylko jeden plik.

Do usuwania plików, używam analogicznie SD.remove() z pominięciem instancji:

SD.remove("123.txt");

Jeżeli tworzymy jakiś rozbudowany projekt to możemy również tworzyć katalogi na karcie przy pomocy funkcji:

plik = SD.mkdir("a/b/c");

Literki a b c oznaczają strukturę katologów, czyli zostanie utworzony folder „c" w folderze „b" i te oba foldery jeszcze w folderze „a". Aby zapisać jakiś plik w takiej strukturze to musimy użyć funkcji *SD.open*:

plik = SD.open("/a/123.txt", FILE_WRITE);
plik = SD.open("/a/b/345.txt", FILE_WRITE);
plik = SD.open("/a/b/c/567.txt", FILE_WRITE);

Gdybyśmy nie poprzedzili ścieżki do pliku przez „/" to by utworzyło plik o takiej nazwie jak wygląda ścieżka.

W taki sam sposób działa usuwanie katalogów, tylko, że zamiast *mkdir* piszemy *rmdir*:

SD.rmdir("a/b/c");

Teraz przejdźmy do operacji na plikach:

Przy pomocy poniższej funkcji możemy zapisywać dane do pliku:

plik.print("wpisywane dane");

plik to instancja klasy *File*. Zasada obsługi *print* jest taka sama jak dla UARTu, czyli *print*– pisanie w jednej linii, *println*– pisanie od nowej linii.

Aby odczytać dane to już nie jest tak łatwo- tzn. nie przy pomocy jednej funkcji ;)

Najpierw musimy sprawdzić czy w danym pliku są jakieś dane (bajty danych) do odczytania przy pomocy:

plik.available()

Następnie gdy jakieś dane będą to musimy wykorzystać funkcję:

plik.read()

Niemniej jednak nie można wpisać tak bezpośrednio tych funkcji, tylko muszą tworzyć logiczną całość, ale o tym później ;)

Okej „przebrnęliśmy” przez podstawowe funkcje. Wydaje się, że jest ich dużo, ale jak zobaczycie sposób ich wykorzystania to się okaże, że obsługa kart jest banalnie prosta ;)

Pierwszy program: Zapis do karty SD

Zaczniemy najpierw od zapisu, ponieważ jest on łatwiejszy.

Przy wykonywaniu operacji na plikach musimy zapamiętać kolejność wykonywania funkcji:

1. Otworzyć/utworzyć plik oraz nadać prawa do zapisu
2. Wstawić dane do pliku
3. Zamknąć plik

A teraz przejdźmy do programu. Program ma zwiększać zmienną o 1, a następnie zapisywać wartość tej zmiennej do pliku.

```
#include <SPI.h>                        //dodaj bibliotekę SPI.h
#include <SD.h>                         //dodaj bilbiotekę SD.h

File plik;

int x = 1;                              //przypisz zmiennej x początkową wartość 1


void setup()
{
 Serial.begin(9600);                    //uruchom UART o prędkości 9600 baud

 Serial.println("Gotowy! (1/3)");
 Serial.println("Szukam karte... (2/3");
 if (!SD.begin(4))                      //sprawdź czy nie ma karty na pinie ChipSelect 4
 {
   Serial.println("Nie wykryto karty(ERR)");          //błąd wykrycia karty
   return;                              //przerwij program
 }

 Serial.println("Karta Wykryta (3/3))");              //Karta wykryta


 if (SD.exists("123.txt"))              //sprawdź czy istnieje plik o nazwie 123.txt
 {

  Serial.println("Plik o podanej nazwie istnieje !");

 }

 else                                   //jeżeli nie istnieje to
 {
```

```
  plik = SD.open("123.txt", FILE_WRITE);              //utwórz plik
  Serial.println("Utworzono plik o nazwie 123.txt");

 }
}


void loop()
{

logger();                                 //wykonaj program zawarty w klasie logger
x = x +1;                                 //dodaj do zmiennej x + 1

}

void logger()
{

plik = SD.open("123.txt", FILE_WRITE);              //otwórz plik 123.txt
plik.println(x);                           //zapisz wartość zmiennej x
plik.close();                             //zamknij/zapisz plik
delay(300);                                //oczekaj 0,3s
Serial.println("Zapisano !");                     //poinformuj o zapisaniu pliku

}
```

Wiem, że może ten program wygląda strasznie, ale zaraz jak go wytłumaczę to powinno się rozjaśnić o wiele więcej. Kurs jak sama nazwa wskazuje ma nauczyć czegoś pożytecznego. W tym przypadku będzie to działanie programu w sposób logiczny oraz chciałbym też pokazać „nowe sposoby" na pisanie programów. Wychodząc z założenia, żeby „Arduino myślało co robi" to zamiast wykonywania programu na ślepo, najpierw niech sprawdzi warunki. Na przykład czy karta jest podłączona, czy utworzony jest plik do którego ma zapisywać itd. Zatem Najpierw sprawdzamy, czy karta jest podłączona do Arduino. Jeżeli nie jest to przerywamy program, bo po co ma być wykonywany skoro najważniejszy warunek nie jest spełniony. Natomiast jak karta została wykryta to sprawdź czy plik do którego chcemy zapisywać istnieje. Jeżeli istnieje to niech wyświetli się komunikat o istnieniu pliku. Jeżeli pliku nie ma, to go stwórz i wyślij komunikat o jego stworzeniu. Teraz przejdźmy do pętli głównej programu. Tutaj mała niespodzianka bo zawiera ona tylko dwie linie kodu. Zatem jak jest on wykonywany ? W tym momencie poznamy klasę.

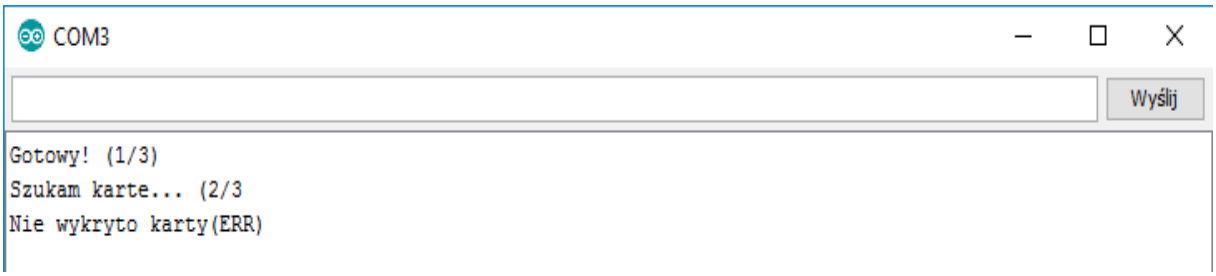
Klasa jest stworzoną przez nas funkcją, której głównym celem jest zwiększenie przejrzystości programu przy czym program, który tworzymy staje się modułowy. Zamiast pisać wszystko w pętli głównej programu to lepiej rozpiszmy tą całą pętlę główną na mniejsze kawałki, do których będzie się tylko odwoływać. W tym przypadku akurat nie ma takiej potrzeby, żeby rozpisywać pętlę główną na mniejsze kawałki, ale to był tylko przykład zastosowania.

Przykład tworzenia klasy i odwoływania się do niej:

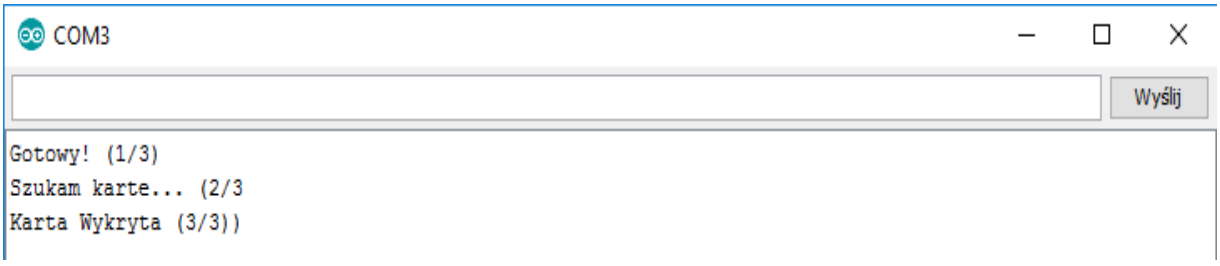
```
void loop()
{

  klasa();
  //dalsza część naszego wspaniałego programu

}

void klasa()
{
   //nasza wspaniała funkcja
 }
```

Wracając teraz do naszego programu. W klasie *logger()* odbywa się cały proces zapisu. Jak widać spełniona jest kolejność operacji na pliku, czyli otwieramy plik z nadaniem praw do zapisu, zapisujemy dane, zamykamy plik. Następnie odczekujemy 0,3 sekundy i wysyłamy komunikat o ukończonym zapisie do pliku. Teraz po zakończeniu naszej klasy program wraca do następnej linii w pętli głównej programu, czyli w tym przypadku zmienna x zostaje zwiększona o 1. Efekt naszego programu powinien być następujący:

1)Brak karty:

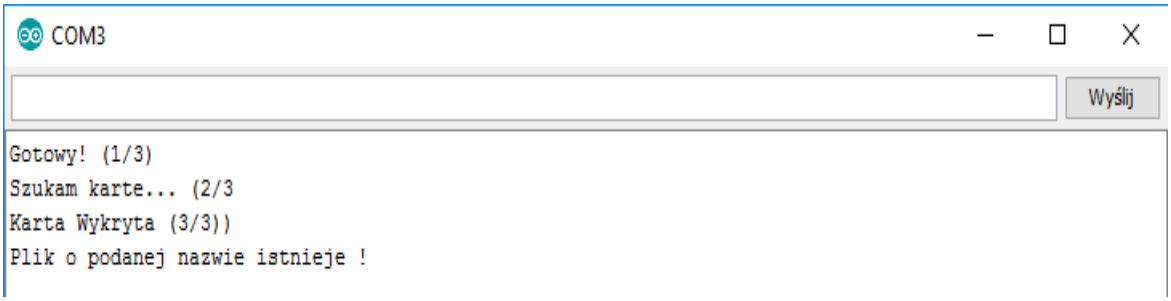


2)Karta wykryta:

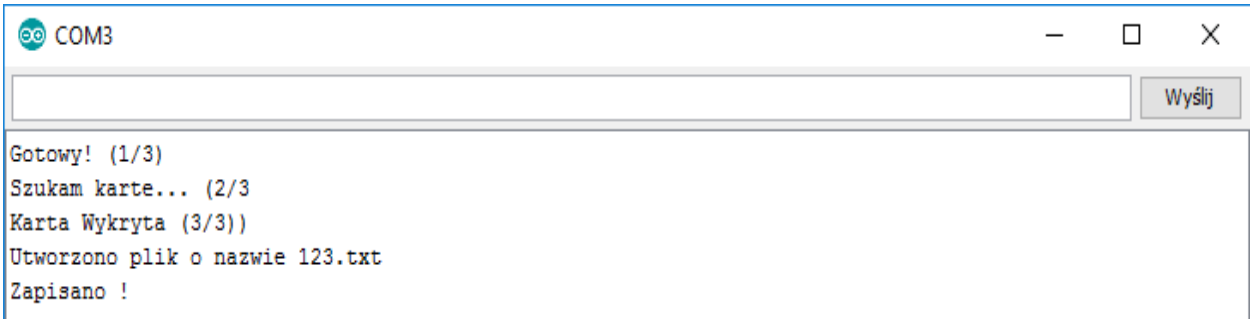


3a) Brak pliku- utworzenie go:

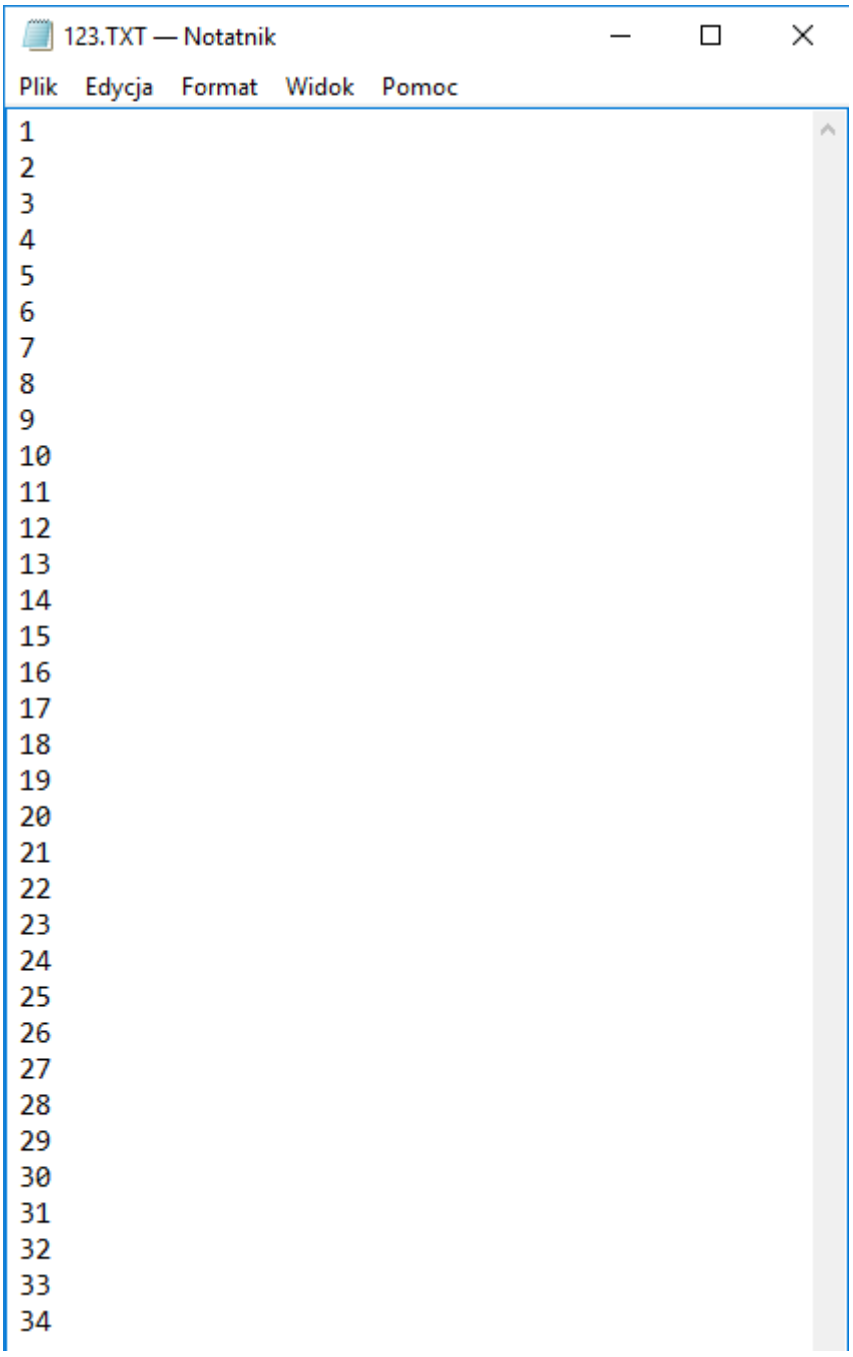
3b) Informacja o istniejącym pliku:



4) Potwierdzenie zapisu do pliku



Po otwarciu utworzonego pliku powinniśmy ujrzeć taki efekt:


123.TXT — Notatnik
Plik Edycja Format Widok Pomoc
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

Drugi program: Odczyt z karty SD

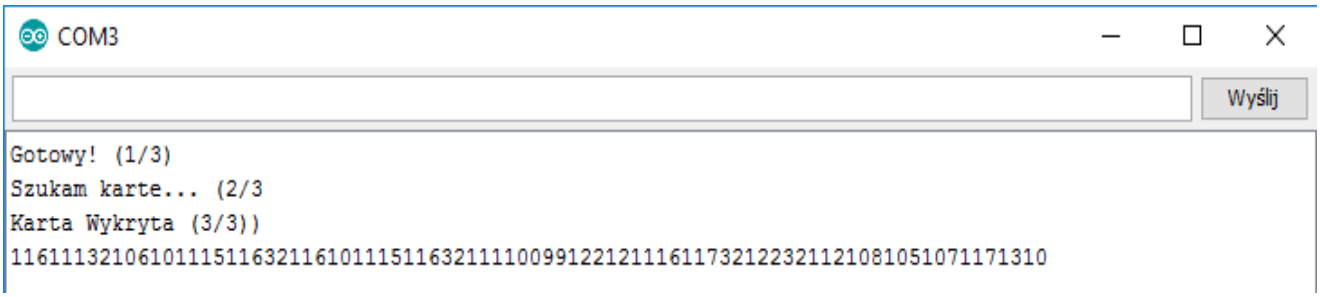
W tym programie zajmiemy się odczytem danych z pliku. Najpierw zapiszemy do niego jakieś dane, a następnie je odczytamy

```
#include <SPI.h>                          //dodaj bibliotekę SPI.h
#include <SD.h>                           //dodaj bilbiotekę SD.h
File plik;
void setup()
{
 Serial.begin(9600);                      //uruchom UART o prędkości 9600 baud
 Serial.println("Gotowy! (1/3)");
 Serial.println("Szukam karte... (2/3");
 if (!SD.begin(4))                        //sprawdź czy nie ma karty na pinie ChipSelect 4
 {
   Serial.println("Nie wykryto karty(ERR)");        //błąd wykrycia karty
   delay(100);
   return;                                //przerwij program
 }
 Serial.println("Karta Wykryta (3/3))");           //Karta wykryta
 plik = SD.open("1234.txt", FILE_WRITE);           //otwórz/utwórz plik 1234.txt
 plik.println("to jest test odczytu z pliku");     //zapisz do pliku
 plik.close();                            //zamknij plik
 delay(1000);                             //odczekaj 1s
 plik = SD.open("1234.txt");              //otwórz plik 1234.txt

   while (plik.available())               //wykonuj pętlę dopóki wszystkie dane
```

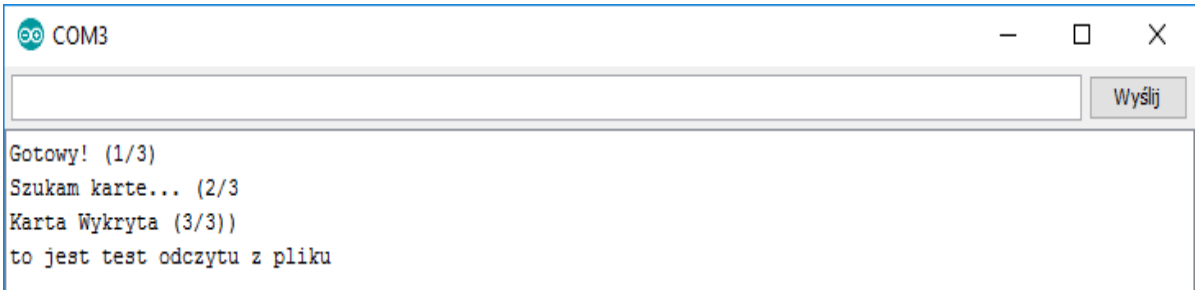
```
  {                                   //nie zostaną sczytane
     Serial.write(plik.read());       //wypisz zawartość pliku
  }
 plik.close();                        //zamknij plik
}
 void loop()
{
 }
```

Początek tego programu jest zbliżony do poprzedniego z wyjątkiem sprawdzenia istnienia pliku. Co ważne program zostanie wykonany tylko raz. Więc po kolei. Tworzymy plik 1234.txt.* Następnie zapisujemy dane do pliku: „to jest test odczytu z pliku”. Zamykamy plik i odczekujemy sekundę. Następnie otwieramy plik i wykonujemy pętlę typu while. Pętla typu while polega na tym, że wykonywana jest tak długo dopóki warunek zawarty w nawiasach nie będzie fałszywy. W tym przypadku pętla będzie wykonywana tak długo, aż nie zostaną żadne dane do przeczytania. Teraz kolejna ważna uwaga wykorzystujemy *Serial.write()*. Gdybyśmy chcieli wykorzystać *Serial.print()* to byśmy otrzymali dane w postaci binarnej. Oczywiście po przekonwertowaniu ich znaki ASCII byśmy otrzymali nasz komunikat, ale po co utrudniać sobie życie skoro można od razu je zapisać jako znaki ASCII.

Serial.print():



Serial.write():



*Przy następnych uruchomieniach programu, plik będzie otwierany, ale nie nadpisywany.

Szkic rejestrujący na karcie SD dane z czujnika

W tej części rozdziału utworzysz prosty szkic rejestrujący w pliku na karcie SD dane z czujnika dołączonego do jednego z wejść analogowych (nr 0).

Rejestrator danych na karcie SD

```
#include <SD.h>          ← Dołączenie biblioteki SD
const int chipSelect = 4;
void setup()
{
Serial.begin(9600);     ← Zainicjowanie portu szeregowego do diagnostyki
Serial.print("Inicjalizacja karty SD...");
pinMode(10, OUTPUT);          ← Ustawienie domyślnego pinu slave select jako wyjścia
if (!SD.begin(chipSelect)) {     ← Sprawdzenie karty SD
Serial.println("Blad karty lub karta niedostepna");
return;
}
Serial.println("Karta zainicjowana.");
}
void loop()
{
        String dataString = "";
        int data = analogRead(0);
        dataString += String(sensor);
        File dataFile = SD.open("datalog.txt", FILE_WRITE);
        if (dataFile) {
                dataFile.println(dataString);
                dataFile.close();
                Serial.println(dataString);
        }
        else {
                Serial.println("Blad otwarcia pliku datalog.txt");
        }
}
```

Rejestrowanie danych nie może już być prostsze. Najpierw musisz sprawdzić, czy karta SD jest dostępna . Jeżeli tak, wszystko jest gotowe do działania, a jeżeli karta nie jest dostępna, następuje powrót z programu.

Po skonfigurowaniu karty można zacząć wykonywać główną pętlę, utworzyć obiekt typu String do przechowywania danych , a następnie odczytać bieżące dane z czujnika i umieścić je w tym obiekcie . Dalej ma miejsce otwarcie pliku, zapisanie w nim danych i zamknięcie . Na potrzeby diagnostyki błędów możesz dane wyświetlić na monitorze portu szeregowego .

Do wejścia analogowego możesz dołączyć dowolny czujnik, na przykład potencjometr, czujnik temperatury lub dalmierz ultradźwiękowy. Gdy dane zostaną zapisane, umieść kartę SD w czytniku swojego komputera i otwórz plik w dowolnym edytorze tekstowym lub środowisku programistycznym. Teraz możesz przedstawić dane w graficznej formie, dzięki której z pewnością lepiej poznasz ich przebieg.