Technologia informacyjna – Arduino

Andrzej Chmielowiec

20 marca 2018

Spis treści

1 Wprowadzenie

Środowisko developerskie dla platformy Arduino korzysta z języka C++, który jest obiektowym rozszerzeniem języka C. Standardowe konstrukcje języka uzupełniono o elementy, które ułatwiają pisanie kodu na mikrokontrolery AVR. Język C++ w przeciwieństwie do większości współcześnie tworzonych języków obiektowych zawiera definicje typów danych. Podstawowymi typami danych są:

`void` – typ określający brak danej,

`char` – typ znakowy (`char c = 'a';`),

`int` – podstawowy typ całkowitoliczbowy reprezentujący zarówno liczby dodatnie, jak też ujemne (`int n = -100;`),

`float` – typ zmiennoprzecinkowy pojedynczej precyzji przeznaczony do reprezentowania liczb rzeczywistych (`float x = 3.1415`).

Dla każdego typu danych możemy zdefiniować wskaźnik (adres) na obszar pamięci przechowujący daną tego typu. Na przykład: `void*`, `char*`, `int*`, `float*` oznaczają adresy do: niezdefiniowanego typu, zmiennej typu znakowego, zmiennej typu całkowitoliczbowego i zmiennej typu zmiennoprzecinkowego.

Dodatkowo każdy typ danych poza `void` może służyć do definiowania tablic – zbiorów elementów tego samego typu. Na przykład:

`char str[3] = {'a', 'b', 'c'};` jest tablicą 3 znaków,

`int t[4] = {-1, 0, 1, 2};` jest tablicą przechowującą cztery liczby całkowite,

`float z[2] = {-1.01, 3.14};` jest tablicą przechowującą dwie liczby zmiennoprzecinkowe.

Środowisko Arduino wprowadza jedną bardzo istotną zmianę względem tego, co określa standard języka C/C++ – wprowadza zamiast funkcji `main` dwie inne funkcje główne:

`setup` – funkcję wywoływaną jednokrotnie po resecie mikrokontrolera,

`loop` – funkcję implementującą główną pętlę programu, czyli funkcję, która wywoływana jest w nieskończonej pętli (po każdym zakończeniu funkcji `loop` wywoływana jest ona po raz kolejny).

W związku z tym pusty projekt w środowisku Arduino ma następującą postać:

```
1 void setup() {
2   // put your setup code here, to run once:
3 }
4
5 void loop() {
6   // put your main code here, to run repeatedly:
7 }
```

Pracę z zestawem Arduino rozpoczniemy od prostego projektu, którego celem będzie napisanie kodu sterującego pracą diody. Przygotujemy prosty program, który będzie co sekundę zapalał i gasił diodę podłączoną do portu mikrokontrolera. W tym celu na poza kodem funkcji definiujemy stałą, która będzie przechowywała identyfikator portu mikrokontrolera, do którego podłączona jest dioda.

```
1 // Podlaczenie diody do portu D3 mikrokontrolera.
2 const int ledPin = 3;
```

Następnie konieczne jest zainicjowanie portu w trybie wyjściowym, co robimy w funkcji `setup` za pomocą następującego polecenia:

```
1 void setup() {
2   // Ustawienie pinu 'ledPin' w trybie wyjsciowym.
```

```
  pinMode(ledPin, OUTPUT);
}
```

W pętli wykonujemy natomiast zapalanie i gaszenie diody, przy czym każda z tych operacji rozdzielona jest odpowiednim czasem oczekiwania.

```
void loop() {
  // Zapalenie diody.
  digitalWrite(ledPin, HIGH);
  // Czekanie 1000[ms] = 1[s].
  delay(1000);
  // Zgaszenie diody.
  digitalWrite(ledPin, LOW);
  // Czekanie 1000[ms] = 1[s].
  delay(1000);
}
```

Kompletny program ma zatem postać:

```
const int ledPin = 3;

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

void loop() {
  digitalWrite(ledPin, HIGH);
  delay(1000);
  digitalWrite(ledPin, LOW);
  delay(1000);
}
```

1.1 Tablice w praktyce

Zmodyfikujemy teraz program służący do migania diodą w taki sposób, aby pokazać praktyczne zastosowanie tablic. W tym celu założymy, że chcemy zdefiniować pewną sekwencję czasów włączenia i wyłączenia diody. Sekwencję tą będziemy przechowywali w tablicy liczb całkowitych. Każdy element tablicy będzie definiował liczbę milisekund opóźnienia do kolejnej zmiany stanu diody. Na przykład tablica:

```
// Stala definiujaca rozmiar tablicy.
const int tsize = 4;
// Zawartosc tablicy.
const int t[4] = {1000, 500, 250, 125};
```

będzie określała 1000 [ms] świecenia diody, 500 [ms] wygaszenia diody, 250 [ms] ponownego świecenia i 125 [ms] ponownego wygaszenia. Pojedynczy cykl zajmie zatem 1875 [ms].

```
void loop() {
  // Petla analizujaca kolejne elementy tablicy.
  for (int i = 0; i < tsize; i++) {
    // Jesli i jest parzyste (i % 2 == 0), to zapal
       diode.
    if (i % 2 == 0) {
      digitalWrite(ledPin, HIGH);
    // W przeciwnym przypadku zgas diode (i jest
       nieparzyste).
    } else {
      digitalWrite(ledPin, LOW);
    }
  }
  // Czekaj odpowiednia liczbe milisekund (odczytana
     z tablicy).
  delay(t[i]);
}
```

W powyższej funkcji zastosowane zostały dwie bardzo ważne konstrukcje języka C/C++: pierwszą z nich jest pętla `for`, a drugą instrukcja warunkowa `if {...} else {...}`. Obie są bardzo często wykorzystywane podczas tworzenia programów, dlatego opiszemy pokrótce ich składnię i działanie. Składnia pętli `for` jest następująca:

```
for (s_1; s_2; s3) {
  s_4;
}
```

W powyższym kodzie

- `s_1` oznacza instrukcję inicjującą działanie pętli. W przypadku naszego programu była to instrukcja `int i = 0`, czyli stworzenie zmiennej całkowitoliczbowej o etykiecie `i` oraz przypisanie jej wartości początkowej równej 0.

– `s_2` warunek kontynuacji pętli. W tym miejscu pojawia się warunek, którego prawdziwość jest weryfikowana przed każdym kolejnym wywołaniem pętli. Jeżeli warunek nie jest spełniony, to program przerywa działanie pętli i przechodzi do wykonania kolejnych instrukcji. W przypadku naszego programu warunkiem kontrolującym działanie pętli był `i < tsize`. Ponieważ `tsize = 4`, więc pętla będzie wykonywana dopóki zmienna `i` będzie miała wartość mniejszą od 4.

– `s_3` oznacza instrukcję, która jest wykonywana po każdym pojedynczym wykonaniu pętli. W naszym przypadku jest to inkrementacja realizowana za pomocą instrukcji `i++`, która po każdym wykonaniu instrukcji `s_4` zwiększa wartość zmiennej `i` o jeden. Przedstawiony kawałek kodu spowoduje więc wykonanie pętli dla wartości `i = 0, 1, 2, 3`.

– `s_4` oznacza instrukcję realizowaną wewnątrz pętli.

Kolejną instrukcją, którą wykorzystaliśmy do realizacji naszej funkcjonalności jest instrukcja warunkowa, której składnia jest następująca:

```
// Pojedyncza instrukcja if.
if (w_1) {
  s_1;
}

// Instrukcja if-else.
if (w_1) {
  s_1;
} else {
  s_2;
}
```

Instrukcja warunkowa określa w jakich warunkach program ma realizować określony wewnątrz instrukcji kawałek kodu. Jeżeli zatem warunek `w_1` jest prawdziwy, to program wykona instrukcję `s_1`, jeżeli natomiast jest nieprawdziwy, to program wykona instrukcję `s_2`. W przypadku naszego programu warunek `(i % 2 == 0)` oznacza sprawdzenie, czy liczba zawarta w zmiennej `i` jest parzysta. Operator `%` wyznacza bowiem resztę z dzielenia liczby `i` przez 2.

Kompletny kod programu wygląda następująco:

```
const int ledPin = 3;

const int tsize = 4;
const int t[4] = {1000, 500, 250, 125};

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

void loop() {
  for (int i = 0; i < tsize; i++) {
    if (i % 2 == 0) {
      digitalWrite(ledPin, HIGH);
    } else {
      digitalWrite(ledPin, LOW);
    }
    delay(t[i]);
  }
}
```

Ćwiczenie 1.1. Napisz program, w którym tablica przechowuje nie tylko czas oczekiwania na koleją instrukcję, ale również przechowuje informację o tym, czy dioda ma być zaświecona, czy nie (zdefiniuje w programie dwie tablice o odpowiednich wartościach).

Ćwiczenie 1.2. Napisz program, który po każdym wykonaniu funkcji `loop` będzie modyfikował czasy znajdujące się w tablicy. UWAGA! Jeśli zawartość tablicy ma być modyfikowana, to tablica nie może być deklarowana jako stała. Użyj `int t[4] = {...};` zamiast `const int t[4] = {...};`.