4. Podstawy programowania (C++)

C++ – język programowania ogólnego przeznaczenia.

Pierwszy krok w stronę programowania wydaje się trudny i taki jest – dla nauczyciela jak i studenta – dla tego krok ten należy wykonywać bardzo ostrożnie. Aby uniknąć trudności technicznych związanych z językami programowania, nasze pierwsze programy napiszemy w pseudokodzie.

Pseudokodem nazywany jest taki sposób zapisu algorytmu, który zachowując strukturę charakterystyczną dla kodu zapisanego w języku programowania ale rezygnuje ze ścisłych reguł składniowych na rzecz prostoty i czytelności.

W celu możliwie największego uproszczenia nauki programowania, ograniczymy się, do kilku komend pseudokodu:

- *Powtarzaj { … }* – pętla, która będzie się wykonywała w kółko tak długo, jak długo będzie spełniony pewien warunek,
- *Jeżeli … to wykonaj { … }* – instrukcja warunkowa, która będzie wykonana tylko wtedy, kiedy spełniony będzie określony warunek. Po tej instrukcji może dodatkowo wystąpić instrukcja *w przeciwnym wypadku {…}*, która będzie wykonywana jeśli poprzedni warunek będzie niespełniony lub *w przeciwnym wypadku, jeżeli … to wykonaj {…}*.
- *Użyj pinu jako …* – jeśli będziemy chcieli użyć któregoś z pinów, musimy ustawić go jako wejście lub wyjście,
- *Odczytaj pin* – pozwoli nam odczytać czy na danym wejściu jest logiczne 0 czy 1,
- *Ustaw pin jako …* – pozwoli na danym wyjściu ustawić 0 albo 1,
- *Zapamiętaj, że … to …* – zmienne, zapamiętujące liczby (podobnie jak markery w PLC),
- *Dodaj, odejmij, pomnóż podziel,*
- *Czekaj ….*

Ponadto użyteczne są też komentarze przy pisaniu bardziej skomplikowanych programów. Każda linijka zaczynająca się od //, # i każdy tekst wewnątrz /* … */ będzie ignorowany.

Cała trudność programowania sprowadza się w tej chwili do przetłumaczenia tego co byśmy chcieli zrobić na pseudokod.

Zadanie 4.1

Zapoznać się z przykładami 1 oraz 2, a następnie napisać pseudokod sygnalizujący za pomocą diody czy została wciśnięta wybrana kombinacja trzech przycisków z pięciu. Dioda podłączona jest do pinu 8, natomiast przyciski do pinów 2, 3, 4, 5 oraz 6. Gdy *wciśnięta* niepoprawna kombinacja, dioda ma się nie świecić. Warunki można łączyć np. *Jeżeli a to 1 oraz b to 1*, to { … }.

Przykład 1

Napisać w pseudokodzie program: zapalaj diodę na 0,5s, potem gaś na sekundę. Dioda podłączona jest do pinu 7. Rozwiązanie:

```
//Żeby sobie ułatwić zadanie i nie musieć zapamiętywać co jest
// gdzie podłączone (teraz mamy tylko jedną diodę podłączoną do
// jakiegoś tam pinu, ale gdyby tych diod było 5, a gdzieś
// jeszcze jakieś inne rzeczy?) powiemy sobie na początek, że
// wszędzie tam, gdzie będziemy używali słowa <<dioda>>, będziemy
// mieli na myśli <<7>>:

      Zapamiętaj, że dioda to 7.

//Z przyczyn technicznych piny nie mogą być w tym samym momencie
// wejściem i wyjściem, trzeba zatem, zanim się ich użyje ustalić
// czy chcemy ich używać jako wejście czy jako wyjście:

      Użyj pinu dioda jako wyjście.

//Jeśli chodzi o kwestie techniczne, to mamy już wszystko opisane
// pozostało nam zatem wykonać samo mruganie.

//Ponieważ chcemy, aby dioda mrugała i nigdy nie przestała, najłatwiej
// będzie wykorzystać do tego pętlę, w której dioda zapali się,
// chwilę poświeci potem zgaśnie i chwilę pobędzie zgaszona:

      Powtarzaj bez końca {
            Ustaw pin dioda na 1.
            Poczekaj 0,5s.
            Ustaw pin dioda na 0.
            Poczekaj 1s.
      }
```

Przykład 2

Napisać w pseudokodzie program: po wciśnięciu przycisku zapal diodę na 3s. Dioda podłączona jest do pinu 7, przycisk do pinu 3. Jeśli przycisk jest wciśnięty, do pinu dociera logiczne 1.

Do rozwiązania tego zadania wykorzystamy część z przykładu 1, a więc:

```
//zapamiętamy w programie do którego pinu podłączona jest dioda
// a do którego przycisk:

      Zapamiętaj, że dioda to 7.
      Zapamiętaj, że przycisk to 3.

//Ustawmy odpowiednio piny jako wyjścia oraz wejścia:

      Użyj pinu dioda jako wyjście.
      Użyj pinu przycisk jako wejście.

//Ponownie, chcemy by program wykonywał się w nieskończoność – tj.
// czekaj na wciśnięcie przycisku, zapal diodę, poczekaj, zgaś
// i czekaj na ponowne wciśnięcie przycisku:

      Powtarzaj bez końca {
            //Odczytajmy i zapamiętajmy czy przycisk jest 1 czy 0:
            Zapamiętaj, że wciśnięcie to {odczytaj pin przycisk}.
            Jeżeli wciśnięcie równa się 1, to wykonaj {
                  Ustaw pin dioda na 1.
                  Poczekaj 3s.
                  Ustaw pin dioda na 0.
            }
      }
```

Krok drugi sprowadza się do przetłumaczenia pseudokodu na język programowania, który później zostanie przetłumaczony (przez komputer) na język maszynowy.

Język wysokiego poziomu (autokod) – typ języka programowania, którego składnia i słowa kluczowe mają maksymalnie ułatwić rozumienie kodu programu dla człowieka, tym samym zwiększając poziom abstrakcji i dystansując się od sprzętowych niuansów.

Pseudokod

```
Powtarzaj bez końca:
      Włącz diodę
      Poczekaj 1 s. (1000 ms.)
      Wyłącz diodę
      Poczekaj 1 s. (1000 ms.)
```

C++ (Arduino)

```
while(1){         //logiczna jedynka
  digitalWrite(dioda, HIGH);
  delay(1000);
  digitalWrite(dioda, LOW);
  delay(1000);
}
```

Arduino składa się z 8-bitowego mikrokontrolera Atmel AVR z uzupełniającymi elementami w celu ułatwienia programowania oraz włączenia innych układów. **Mikrokontroler**, mikrokomputer jednoukładowy to scalony system mikroprocesorowy, zrealizowany w postaci pojedynczego układu zawierającego procesor, pamięć RAM, układy wejścia-wyjścia i na ogół pamięć programu.

Program z przykładu 1 napisany dla Arduino:

```
int dioda = 7;                // definicja zmiennej (globalnej)

void setup() {                //funkcja wykonywana raz, po włączeniu zasilania
  pinMode(dioda, OUTPUT);     //ustawia pin nr 7 jako wyjście
}

void loop() {                 //funkcja wykonywana w kółko
  digitalWrite(dioda, HIGH);  //ustaw na 7 wyjściu 5V (logiczne 1)
  delay(500);                 //czekaj 500 ms. = 0.5 s.
  digitalWrite(dioda, LOW);   //ustaw na 7 wyjściu 0V (logiczne 0)
  delay(1000);                //czekaj 1 s.
}
```

Jak widać większość instrukcji i całość programu jest bardzo podobna do rozwiązania przykładu pierwszego. Należy się jednak kilka uwag:

- Wszystko co jest wewnątrz nawiasów klamrowych {} funkcji *void loop()* będzie wykonywane w kółko po uruchomieniu Arduino,
- wszystko co jest wewnątrz {} funkcji *setup*, wykonane zostanie raz po uruchomieniu Arduino. Jest to doskonałe miejsce na np. ustawienie pinów jako wejścia/wyjścia czy uruchomienie komunikacji RS (o tym kiedy indziej).
- deklaracje zmiennych, aby nie komplikować sobie nauki, będziemy umieszczać na początku programu czyli przed *setup* i *loop*, a wszystkie zmienne będą typu *int*, chyba, że w poleceniu będzie zaznaczone co innego. Zmienna int może przyjmować wartości od -32768 do 32767 (tylko całkowite). Deklaracja ma postać:

int	dioda	= 7;
typ zmiennej	nazwa zmiennej	przypisanie wartości (opcjonalne)

Tłumaczenie z pseudokodu (programowanie w Arduino)

Aby nasze programy mogły zadziałać musimy je przetłumaczyć na język

Zacznijmy więc od *szablonu* pustego programu w którym będziemy umieszczali nasze *tłumaczenia*:

```
// miejsce na deklaracje zmiennych
// jeśli gdzieś będziemy używali zmiennych (np. do zapamiętania odczytanej
// wartości na pinie wejściowym), należy o tym poinformować program przed
// void setup(){}. Np.: int wcisniecie1;
// albo int przycisk3 = 4;

void setup() {
                              // funkcja wykonywana raz, po włączeniu zasilania
                              // Tutaj umieścimy rzeczy, które mogą wykonać się
                              // tylko raz, ponieważ później nie ulegną zmianie.
                              // Np. pinMode(…).
}

void loop() {
                              // funkcja wykonywana w kółko (bez końca)
                              // Tutaj znajdą się rzeczy, które program ma
                              // wykonywać w sposób ciągły (np. sprawdzanie co
                              // jest na wejściach i ustawianie co ma być na
                              // wyjściach)
}
```

Bardzo często przy programowaniu pojawiać się będzie słowo **WARUNEK**. Warunek jest to pewne działanie logiczne lub arytmetyczne zwracające wartość 0 (warunek niespełniony) lub inną (warunek spełniony):

a > b lub b < a	*a* większe od *b*
a >= b	*a* większe bądź równe *b*
a == b	*a* równe *b (ważne, podwójne równa się!)*
a != b	*a* nie-równe b

Łączenie warunków: ORAZ: if (*warunek1* && *warunek2*){...}

LUB: if (*warunek1* || *warunek2*){...}

Poniżej *tłumaczenia* wcześniejszych instrukcji

Należy pamiętać o tym, że każda instrukcja (poza pętlami i funkcjami warunkowymi) musi się kończyć średnikiem (;).

Jeżeli … to wykonaj { … } – instrukcja warunkowa, która będzie wykonana tylko wtedy, kiedy spełniony będzie określony warunek, np.:

```
zmienna = 10;
if(zmienna == 11){
      digitalWrite(dioda, HIGH);
}
```

Co należy odczytać jako: ustaw zmienną na 10, następnie sprawdź czy zmienna ma wartość 11. Jeśli tak – zapal diodę, jeśli nie – pomiń.

Użyj pinu jako … – jeśli będziemy chcieli użyć któregoś z pinów, musimy ustawić go jako wejście lub wyjście.

```
pinMode(pin, OUTPUT);   //ustaw pin jako wyjście (np. dioda)
pinMode(pin, INPUT);    //ustaw pin jako wejście (np. przycisk)
```

Odczytaj pin – pozwoli nam odczytać czy na danym wejściu jest logiczne 0 czy 1,

```
digitalRead(10);  //sprawdza czy pinie nr 10 jest logiczne 0 czy 1
```

Samo odczytanie nie miałoby większego sensu, ale można wynik funkcji zapisać do zmiennej albo wykorzystać jako WARUNEK funkcji *if*, *while* itd.:

```
//zadeklaruj zmienną wcisniety oraz zmienną gdzie podpięty jest przycisk
int wcisniety = 0;
int przycisk = 7;

//zapamiętaj czy przycisk jest wciśnięty czy nie
wcisniety = digitalRead(przycisk);

//jeżeli przycisk był wciśnięty – zapal diodę
if(wcisniety == 1){
      digitalWrite(dioda, HIGH);
}

//jeżeli przycisk nie był wciśnięty – zgaś diodę
if(wcisniety == 0){
      digitalWrite(dioda, LOW);
}
```

Ustaw pin jako … – pozwoli na danym wyjściu ustawić 0 albo 1:

```
digitalWrite(13, HIGH);       //ustawia na pinie 13 logiczne 1
digitalWrite(13, LOW);        //ustawia na pinie 13 logiczne 0
```

Zapamiętaj, że … to … – zmienne, zapamiętujące liczby (znaki):

```
zmienna = 12;  //aby zapamiętać nową wartość zmiennej, używamy znaku
               // (pojedynczego!) równa się. Od tego momentu zmienna
               // zmienna będzie przechowywała wartość 12.
```

Powtarzaj { … } – pętla, która będzie wykonywana tak długo, jak długo będzie spełniony warunek. Jedną z takich pętli jest pętla *while(warunek){ciało_funkcji}, np.*:

```
zmienna = 0;
while(zmienna < 10){
      digitalWrite(dioda, HIGH);
      delay(1000);
      digitalWrite(dioda, LOW);
      delay(1000);
      zmienna = zmienna + 1;
}
```

Co można odczytać, jako: ustaw zmienną na 0, następnie sprawdź czy zmienna jest mniejsza od 10 – jeśli tak, zapal diodę i poczekaj sekundę, zgaś diodę i poczekaj sekundę. Zwiększ zmienną o jeden. Koniec pętli – wróć na początek i ponownie: sprawdź czy zmienna jest mniejsza od 10 itd… Jeśli warunek nie jest spełniony, program *przeskakuje* za klamrę }.

Dodaj, ***odejmij***, ***pomnóż podziel***:

```
zmienna = 3 + 2;        //zmienna przyjmie wartość 5
zmienna = 3 – 2;        //zmienna przyjmie wartość 1
zmienna = 3 * 2;        //zmienna przyjmie wartość 6
zmienna = 3 / 2;        //zmienna przyjmie wartość 1!
                        // jeśli zmienna była typu int, może przechowywać
                        // wartości jedynie całkowite, a komputer
                        // zaokrągla tylko w dół

zmienna++;              //zwiększ zmienną o 1
zmienna--;              //zmniejsz zmienną o 1
```

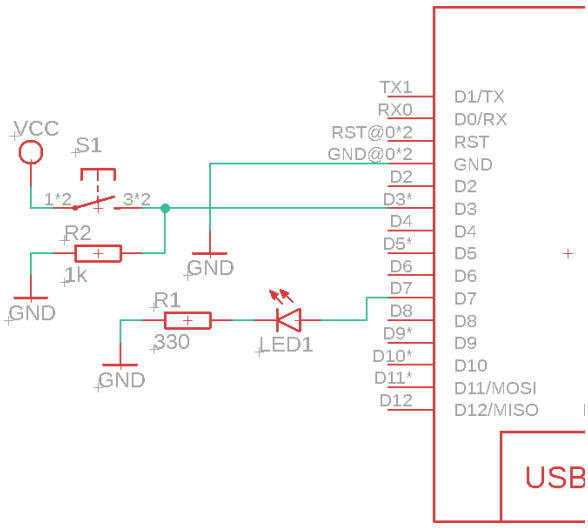
Czekaj…:

```
delay(1000);            //program zatrzyma się na 1s. 1s = 1000ms
```

Zadanie 4.2

Przetłumaczyć na C++ przykład 2 oraz przetestować jego działanie (patrz koniec instrukcji).

Układ do testowania programu z przykładu 2 przedstawiono po prawej. VCC to wyjście *5V* na Arduino.



Zadanie 4.3

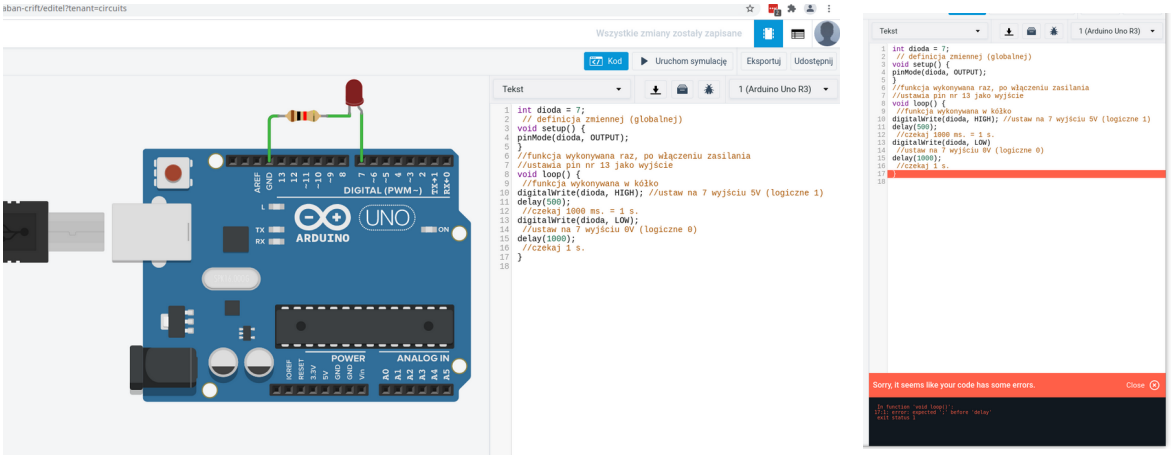
Przetłumaczyć na C++ zadanie 5.1 (bez testowania)

Zadanie dodatkowe

Napisać program w pseudokodzie a następnie w C++ sterujący silnikiem. Do sterowania służą dwa przyciski (lewo oraz prawo podłączone do pinów 6 oraz 7). Silnik podłączony jest jest do pinów 8 i 9. Ustawienie wyjść na 1 i 0 powoduje ruch silnika w lewo, ustawienie wyjść na 0 i 1 powoduje ruch silnika w prawo, natomiast 0 i 0 – zatrzymanie silnika.

Sprawdzanie programów na Arduino

Aby sprawdzić poprawność napisanych przez nas programów, możemy ponownie posłużyć się stroną tinkdercad.com. Program wgrać można na Arduino, musimy więc go użyć w naszym układzie:



Po wybraniu K*od* z menu oraz przestawieniu (poniżej) z *Bloki* na *Tekst* pojawi nam się miejsce na napisany przez nas kod. Po wybraniu *uruchom symulację*, program powinien się skompilować i uruchomić. W przypadku błędów, dostaniemy o tym informację.

Należy jednak pamiętać, że żeby przycisk *dawał* sygnał, musi być gdzieś do Arduino podłączony, podobnie jak diody – żeby się zaświecić, muszą istnieć w symulacji.

Patryk Król

https://www.arduino.cc/reference/en/