Bascom czy C?

część 2

W drugiej części artykułu kontynuujemy porównanie dwóch bardzo popularnych języków programowania mikrokontrolerów: Bascoma i C/C++. Nie odpowiadamy na pytanie: który z nich jest lepszy? Ocenę i wybór pozostawiamy Czytelnikom.

Operacje arytmetyczno-logiczne

Nadeszła pora na operacje arytmetyczne i logiczne. W Basicu równania są ograniczone do postaci:

```
A = B + C
```

Nie można budować bardziej skomplikowanych formuł, takich jak:

```
A = B + C + 1
```

Występuje więc konieczność rozbijania bardziej złożonych operacji na krótsze:

```
A = B + C
A = A + 1
```

W **tab. 1** przedstawiono zestawienie operacji arytmetyczno-logicznych dla języków Basic i C.

C wprowadza ponadto operatory, które nie mają bezpośrednich odpowiedników wśród operatorów znanych z Basica, jak i wielu innych języków:

`>>` przesunięcie bitów w prawo, np.: `A = B>>2;` oznacza: „przesuń B o 2 bity w prawo i przypisz do A”

`<<` przesunięcie bitów w lewo (analogicznie)

`++` oznacza „zwiększ o jeden” (inkrementuj) – może być użyty przed nazwą zmiennej:

`A = ++B;` „zwiększ B o jeden i przypisz tę wartość A”

a także po zmiennej:

`A = B++;` „A przypisz B i dopiero teraz zwiększ B o jeden”

`--` oznacza „zmniejsz o jeden” (dekrementuj) – używany tak jak `++`.

Zarówno `++` jak i `--` mogą funkcjonować jako zupełnie osobne instrukcje, wtedy nie ma znaczenia, czy występują przed, czy po zmiennej, np.: `++D; D++;`

W obu przypadkach będzie to równoznaczne Basicowemu `Incr D`.

Język C wprowadza także następujące operatory: `+=, -=, *=, /=, %=, <<=, >>=;`

Dzięki nim wyrażenia takie jak:

```
A = A + B;
A = A / B;
A = A<<B;
```

można zapisać w krótszy sposób:

```
A += B;
A /= B;
A <<= B;
```

W C nie ma ograniczeń w długości operacji arytmetycznych. Kolejność wykonywania działań jest taka, jak ogólnie przyjęta w matematyce:

```
A = 2+2*2; /* spowoduje przypisanie
A wartości 6 */
```

Jeśli zachodzi potrzeba, kolejność wykonywania działań można zmienić stosując nawiasy:

```
A = (2+2)*2; /* czyli A przypisz 8 */
```

Równania można rozbudowywać do bardzo złożonych form, „wplatać” w nie odniesienia do zmiennych wskazywanych przez wskaźniki, wartości zwracane przez funkcje, a nawet w samym środku równania zmieniać zawartość poszczególnych zmiennych:

```
A = 10;
B = 20;
C = 50;
D = 1;
A-= (D=B-C)*(C+3)/sin(B);
```

W ostatnim przypadku równanie zostanie rozłożone podczas kompilacji i wykonane później w kolejności:

```
D = B-C;
A = A - D*(C+3)/sin(B);
```

Charakterystyczna dla języka C jest też np. poniższa konstrukcja:

```
A=B=C=D=3;
```

czyli przypisanie zmiennym `A, B, C, D` wartości 3. Inną, specyficzną konstrukcją, którą można stosować m.in. właśnie w równaniach, jest:

```
C = (warunek)?A:B;
```

Polega ona na tym, że jeśli warunek jest prawdziwy, to prawa strona przyjmuje wartość `A`, a jeżeli jest nieprawdziwa – wartość `B`. Przykład:

```
A = (B<=18)?B:25;
```

W tym przypadku zmiennej `A` zostanie przypisanie wartość `B`, jeśli `B` będzie mniejsze lub równe 18, w przeciwnym wypadku `A` zostanie przypisane 25. Zamiast `B` i 25 można wstawiać dowolnie długie wyrażenia, a samej konstrukcji `(warunek)?A:B` można używać w równaniach jak normalnej zmiennej np.:

```
C = A*((D>=0)?D:(-D));
```

Konstrukcja ta nie dotyczy tylko liczb. Dość specyficznym jej wykorzystaniem może być np.:

```
printf((udalo_sie)?(„Udało się!”):
(„Nie udało się.”));
```

Tab. 1. Zestawienie operacji arytmetyczno-logicznych dla języków Basic i C

Basic		C	
Operatory arytmetyczne:			
+	Suma	+	Suma
-	Różnica	-	Różnica
*	Iloczyn	*	Iloczyn
/	Iloraz	/	Iloraz
\	Iloraz	\	Iloraz
^	Potęga		
MOD	Modulo (reszta z dzielenia)	%	Modulo (reszta z dzielenia) potęga
Operatory logiczne:			
NOT	Negacja	~	Negacja (dla każdego bitu osobno)
		!	Negacja (tylko dla najmłodszego bitu)
OR	Suma	|	Logiczne OR (porównywany jest każdy bit obu zmiennych)
AND	Iloczyn	&	Logiczne AND (porównywany jest każdy bit obu zmiennych)
XOR	Exclusive OR	^	Logiczne XOR (porównywany jest każdy bit obu zmiennych)
		||	Logiczne OR (porównywany jest tylko najmłodszy bit obu zmiennych)
		&&	Logiczne AND (porównywany jest tylko najmłodszy bit obu zmiennych)
Operatory porównujące dwie wartości:			
=	Równe	==	Równe (często mylony z =, który jest znakiem przypisania)
<>	Nierówne	!=	Nierówne
<	Mniejsze	<	Mniejsze
>	Większe	>	Większe
<=	Mniejsze lub równe	<=	Mniejsze lub równe
>=	Większe lub równe	>=	Większe lub równe

Reprezentacja liczb

Przy okazji wykonywania działań należałoby zapoznać się ze sposobem zapisu liczb w obu językach. Liczby dziesiętne są reprezentowane identycznie: 236 to dwieście trzydzieści sześć. Jednak bardzo często, szczególnie wśród programistów mikrokontrolerów wykorzystywany jest zapis szesnastkowy. W Basicu liczba 255 zapisana w systemie heksadecymalnym to

&HFF. Przedrostek &H oznacza właśnie ten system liczbowy. W języku C sprawa ma się w zasadzie identycznie, tyle że przedrostkiem jest 0x, czyli 255 zapiszemy w postaci 0xFF. W Basicu można także zapisywać liczby w systemie binarnym, korzystając z przedrostka &B: 255 to &B11111111. Język C nie obsługuje standardowo takiego zapisu liczb, choć niektóre kompilatory także tu wprowadzają swoje rozszerzenia, np. 255 zapiszemy jako 0b11111111. W specyfikacji języka C zawarto natomiast obsługę systemu ósemkowego – wystarczy jako pierwszy znak podać 0 (zero). Wówczas 0100 oznacza 64 (w systemie dziesiętnym), a nie 100 lub 4.

Organizacja kodu – funkcje, procedury

W obu językach występują mechanizmy umożliwiające blokową organizację kodu. W Basicu są to funkcje i procedury. W C odpowiednikiem procedury jest funkcja niezwracająca żadnej wartości. W obu językach przed użyciem funkcji/procedury konieczna jest jej wcześniejsza deklaracja. W Basicu procedurę deklarujemy tak:

```
Declare Sub nazwa_procedury(para-
metr1 As typ1, parametr2 As typ2)
```

oraz funkcję:

```
Declare Function nazwa_funckcji(pa-
rametr1 As typ1, parametr2
As typ2) As typ_funkcji
```

Parametry są opcjonalne – może ich w ogóle nie być. Przed nazwą każdego z parametrów można napisać słowo kluczowe `byval` albo `byref`. `Byref` jest dodawany domyślnie, jeżeli nie podamy `byval`. `Byval` oznacza, że z wnętrza funkcji/procedury zmienna jest widziana jako kopia tego, co podaliśmy przy wywołaniu. Jeżeli parametr został zadeklarowany z użyciem `byref`, to można zmienić z wnętrza funkcji wartość zmiennej, która została podana jako parametr przy wywołaniu. Oto przykłady deklaracji ciała funkcji i procedury:

```
Function Kwadrat1(Liczba As Integer)
As Integer
Kwadrat1 = Liczba * Liczba
End Function

Sub Kwadrat2(Liczba As Integer)
Liczba = Liczba * Liczba
End Sub

Sub Kwadrat3(Byval Liczba As Inte-
ger)
Liczba = Liczba * Liczba
End Sub
```

Jeżeli wywołamy procedurę `Kwadrat3` podając jako parametr zmienną `A` równą 10, to po wyjściu z tej procedury zmienna `A` nadal będzie miała wartość 10. Aby zmienić jej wartość na 100, należy wywołać procedurę `Kwadrat2`.

Zakończenie procedury i funkcji jest sygnalizowane odpowiednio:

```
End Sub
```

lub

```
End Function
```

a także przy wywołaniu

```
Exit Sub
```

lub

```
Exit Function
```

Aby określić, jaką wartość zwróci funkcja, posługujemy się przypisaniem:

```
nazwa_funkcji = wartosc
```

Jest ono dość niepraktyczne, jeżeli chcemy zmienić nazwę funkcji. Jeżeli się tego przypisania nie zrobi, funkcja zwróci wartość domyślną, np. 0 dla typów liczbowych, tekst o zerowej długości dla typu `String`.

W C zdefiniowanie ciała funkcji jest jednocześnie jej deklaracją i wygląda tak:

```
typ_funkcji nazwa_funkcji(typ1 para-
metr1, typ2 parametr2)
{
/*tu jest kod funkcji*/
}
```

W przypadku funkcji niezwracającej żadnej wartości:

```
void nazwa_funkcji(typ1 parametr1,
typ2 parametr2)
{
/*tu jest kod funkcji*/
}
```

Dozwolone jest deklarowanie funkcji, które nie mają z góry określonej liczby parametrów:

```
int srednia(int a, ...)
{
}
```

Tak zadeklarowaną funkcję można teraz wywołać z różną liczbą parametrów:

```
s = srednia(10, 44, 23, c, e);
r = srednia(d, e);
```

Dodatkowe parametry nie muszą być tego samego typu. Opis sposobu, w jakim uzyskuje się do nich dostęp, wykracza poza ramy tego artykułu.

W C parametry zawsze są widziane z wnętrza funkcji jako kopia tego, co podaliśmy przy wywołaniu. Dopiero C++ wprowadza odpowiednik `byref` z Basica. Można także wybrać sposób, w jaki będą one przekazywane funkcji – albo przez stos, albo przez rejestry. Szczegóły zależą od kompilatora. Programista może zadeklarować wcześniej funkcję, ale nie jest to konieczne. Jeżeli chcemy, by funkcja mogła zmienić zmienne przekazywane jej jako parametr, należy po prostu przekazać wskaźnik, a nie zmienną. Funkcję deklarujemy np. tak:

```
void kwadrat(int *zmienna)
{
   *zmienna = (*zmienna) * (*zmienna);
}
```

Aby zmienić zmienną:

```
int a = 10;
```

wywołujemy:

```
kwadrat(&a);
```

Zwracanie wartości i jednocześnie powrót z funkcji następuje po wykonaniu polecenia

```
return wartosc;
```

Na przykład:

```
int kwadrat(int zmienna)
{
   return zmienna*zmienna;
}
```

Funkcję tę wywołamy w poniższy sposób:

```
a = kwadrat(a);
```

Wyjście z funkcji typu `void` (czyli odpowiednik procedury) odbywa się poprzez instrukcję `return` lub bez niej.

Parametrami funkcji mogą być zmienne (także typów strukturalnych oraz unie, ale o tym będzie mowa w dalszej części artykułu), stałe, wskaźniki oraz całe wyrażenia, włączając w to także wywołania do innych funkcji, np.:

```
OutPort((A+C)*(d-a)
+sqrt(sin(45*PI/180));
```

Wywołanie procedury w Basicu jest inne niż funkcji w C będącej odpowiednikiem procedury.

W Basicu zapisujemy:

```
Call Procedura()
```

lub

```
Call Procedura
```

W języku C natomiast:

```
Procedura();
```

W C funkcje są zawsze oddzielone od głównego programu, mogą się znajdować przed i za funkcją `main`. Jednak nigdy jedna funkcja nie może znaleźć się wewnątrz drugiej ani wewnątrz jakiegokolwiek innego segmentu. Dzięki temu dopóki nie wywołamy danej funkcji, nie zostanie ona wykonana. W Basicu jest inaczej – przykładem niech będzie ten oto program i jego zmodyfikowana wersja:

```
Declare Sub Proc

Print „Program główny”
Call Proc
End

Sub Proc
Print „Procedura”
End Sub
```

Jego wynikiem będzie wysłanie do portu szeregowego tekstu „Program główny”, a następnie (w rezultacie wywołania procedury) – tekstu „Procedura”. Procedura `Proc` jest umieszczona za słowem kluczowym `End`, oznaczającym koniec programu. Zmieńmy teraz jej położenie:

```
Declare Sub Proc
Sub Proc
Print „Procedura”
End Sub
Print „Program główny”
Call Proc
End
```

Jedyne, co zrobi ten program, będzie wysłanie tekstu „Procedura”, po czym nastąpi koniec programu. W C można funkcje rozmieszczać w dowolnej kolejności:

```
void Proc()
{
   printf(„Procedura”);
}
int main()
{
   printf(„Program główny”);
   Proc();
}
```

Powyższy program wypisze najpierw tekst „Program główny”, a następnie „Procedura”. Jeżeli chcemy funkcję `Proc` umieścić za funkcją `main`, należy ją wcześniej zadeklarować, żeby po dojściu do wywołania `Proc()` kompilator wiedział, co to jest `Proc`:

```
void Proc(); /* to jest deklaracja
procedury „Proc” */
int main()
{
   printf(„Program główny”);
   Proc();
}

void Proc()
{
   printf(„Procedura”);
}
```

Ogromną zaletą funkcji jest to, że można dzięki nim skrócić kod wynikowy, gdy często korzystamy z powtarzającego się fragmentu programu. Ich wywołanie na wielu procesorach i mikrokontrolerach jest nieznacznie wolniejsze od instrukcji skoku (np. `goto` lub asemblerowe `jmp`, itp.) W C można zadeklarować funkcję z użyciem słowa kluczowego `inline`, co powoduje, że przy każdym jej wywołaniu tak naprawdę nie jest wykonywany skok do podprogramu, lecz w dane miejsce jest wstawiany kod funkcji. Takie rozwiązanie jest szybsze od wywołania, ale powoduje zwiększenie objętości kodu wynikowego.

Instrukcje warunkowe

Większość zadań realizowanych przez każdy program następuje po spełnieniu odpowiednich warunków. Ich sprawdzanie może się odbywać różnymi sposobami. W poszczególnych językach programowania składnia instrukcji warunkowych może być odmienna. Zobaczmy jak są one zrealizowane w omawianych językach. Najpopularniejszą instrukcją warunkową jest `if` – zarówno w Basicu, jak i w C. W tym pierwszym języku wygląda ona następująco:

```
If warunek Then
  'ten kod jest wykonywany,
  'jeżeli warunek jest prawdziwy
Endif
```

Można dodać drugi blok, który jest wykonywany, gdy warunek nie jest prawdziwy:

```
If warunek Then
  'ten kod jest wykonywany,
  'jeżeli warunek jest prawdziwy
Else
  'ten kod jest wykonywany,
  'jeżeli warunek nie jest prawdziwy
Endif
```

Konstrukcję tę można także rozbudowywać o dodatkowe bloki `Elseif`:

```
If warunek1 Then
  'ten kod jest wykonywany,
  'jeżeli warunek1 jest prawdziwy
Elseif warunek2 then
  'ten kod jest wykonywany,
  'jeżeli warunek1 nie jest
  'prawdziwy, a prawdziwy jest
  'warunek2
Else
  'ten kod jest wykonywany,
  'jeżeli ani warunek1,
  'ani warunek2 nie jest prawdziwy
Endif
```

Można więc tworzyć w ten sposób bardzo złożone instrukcje warunkowe. Badanie warunku w języku C wygląda podobnie:

```
if (warunek)
  instrukcja;
```

Gdy chcemy wykonać więcej niż jedną instrukcję, należy objąć je nawiasami klamrowymi:

```
if (warunek)
{
  instrukcja1;
  instrukcja2;
}
```

Rozmieszczenie nawiasów nie ma znaczenia – ta sama instrukcja może być zapisana na wiele sposobów (dotyczy to nawiasów w ogóle, nie tylko przy instrukcji `if`), np.:

```
if (warunek){
   instrukcja1;
   instrukcja2;
}

if (warunek){instrukcja1; instrukcja2;}
```

Można także zadeklarować blok `else`:

```
if (warunek)
   instrukcja1;
else
   instrukcja2;
```

Instrukcją może być dowolna instrukcja – także instrukcja `if` – można więc budować na przykład poniższe konstrukcje:

```
if (warunek1)
   instrukcja1;
else
   if (warunek2)
       instrukcja2;
   else
       instrukcja3;
```

Dlatego zrezygnowano z osobnej instrukcji `elseif`.

W Basicu warunki nie mogą być skomplikowane – dozwolone są tylko proste porównania i ich łączenie za pomocą operatorów logicznych, np.:

```
if 10>a and a>4 then
   instrukcja
endif
```

W C warunkiem może być w zasadzie każde wyrażenie i zmienna. Wartość „prawda” będzie miało wyrażenie, którego wartość nie równa się 0. Należy więc uważać, gdy np. mamy zmienną typu `char` (ze znakiem) i przypiszemy jej wartość np. -1. -1 nie jest równe 0, więc warunek zostanie potraktowany jako prawdziwy:

```
if (-1)
   to_polecenie_zostanie_wykonane;
```

Warunek mogą także stanowić skomplikowane wyrażenia (łączące zarówno operatory logiczne, jak i arytmetyczne):

```
if (((A*A+sin(B))>=((c)?10:2)) && (!end))
{
}
```

Inną często używaną instrukcją w Basicu jest `Select` i jej odpowiednik `switch` w języku C. W Basicu zapisujemy:

```
Select Case zmienna
  Case test1: instrukcje
  Case test2: instrukcje
  ...
  Case testn: instrukcje
  Case Else: instrukcje
End Select
```

Jako `test1`, `test2` itd. wpisujemy konkretną wartość, zakres wartości albo test, jaki mikrokontroler ma przeprowadzić, np.:

```
Select Case w
  Case 4: Print „w = 4”
  Case 7 To 10: Print „w jest w
     przedziale od 7 do 10”
  Case Is > 100: Print „w jest
     większe od 100”
  Case Else: Print „żaden z powyższych
     warunków nie został spełniony”
End Select
```

W C mamy instrukcję `switch`:

```
switch(w)
{
   case wartosc1: instrukcje;
   case wartosc2: instrukcje;
   default: instrukcje;
}
```

Jest to złożona instrukcja warunkowa, która wykonuje skok w różne miejsca, kierując się konkretnymi wartościami badanej zmiennej. Nie można przeprowadzać tu testów, takich jak `7 To 10`. Jej działanie jest także inne – przypomina ona skok do etykiety. Rozważmy przykład:

```
switch(w)
{
   case 3:
   case 4:
   case 5: instrukcja1;
             break;
   case 7: instrukcja2;
   case 8: instrukcja3;
             break;
   default: instrukcja4;
}
```

`case 3`, `case 4` itd. można traktować jak etykiety. W tym przypadku `case 3`, `case 4` i `case 5` są etykietami wskazującymi bezpośrednio na `instrukcja1`. Dlatego jeżeli zmienna `w` będzie równa 3, 4 lub 5, zostanie wykonany skok do `instrukcja1`. Zostanie ona wykonana, po czym program dotrze do instrukcji `break`, co spowoduje zakończenie działania instrukcji `switch`. Jeżeli w=7, to nastąpi skok do `instrukcja2`. Zostanie więc wykonana `instrukcja2` i `instrukcja3`, a następnie `break;` czyli zakończenie instrukcji `switch`. Jeżeli w=8, to wykonana zostanie tylko `instrukcja3` i `break`. Jeżeli wartość zmiennej `w` nie pasuje do żadnego podanego wyżej warunku, to zostanie wykonany skok do etykiety `default`. Nastąpi wykonanie `instrukcja4` i wyjście z instrukcji `switch`.

Kuba Klimkiewicz