Podstawy języka C

Część 2

W drugim odcinku kontynuujemy poznawanie podstaw języka C. Ćwiczenia wykonujemy z wykorzystaniem opisanego przed miesiącem programu Visual Studio.
Nie zrażaj się, gdy czegoś nie zrozumiesz – wykonaj podane ćwiczenia.

Operatory

W języku C występuje wiele operatorów: arytmetyczne, porównania, logiczne, bitowe i inne. Przyjrzymy się najważniejszym z nich. Jak widać w **tabeli 3**, przypisanie odbywa się za pomocą znaku równości. Należy pamiętać, że pojedynczy znak równości nie służy do wykonywania porównań. Operatorem równości jest podwójny znak równości (==). Dodawanie i odejmowanie wykonujemy za pomocą znaków + i – znanych z matematyki. Operatorem mnożenia jest gwiazdka, a operatorem dzielenia jest ukośnik. Znak procentu służy do wykonywania operacji modulo, czyli reszty z dzielenia a przez b. Język C ma także operatory do inkrementacji i dekrementacji, czyli zwiększania o 1 i zmniejszania o 1. Mają one postać podwójnego znaku plus i podwójnego znaku minus. Operator ten może znajdować się przed zmienną, której dotyczy, lub po. Różnica dotyczy momentu zadziałania operatora względem momentu odczytania zawartości. Uruchom w Visual Studio następujący fragment kodu:

```
int a = 5;
printf("%d\n", a++);
printf("%d\n", a++);
```

Zostaną wyświetlone wartości 5 oraz 6. A teraz spróbuj tak:

```
int a = 5;
printf("%d\n", ++a);
printf("%d\n", ++a);
```

W okienku konsoli zobaczysz wartości 6 oraz 7. Dlaczego? W pierwszym przypadku wartość zmiennej a najpierw jest przekazywana jako argument do funkcji printf, a dopiero potem jest zwiększana. W drugim przypadku zwiększenie wartości zmiennej jest wykonywane, zanim trafi ona do funkcji wyświetlającej. Jeśli inkrementację lub dekrementację wykonalibyśmy wcześniej, umiejscowienie operatora nie miałoby znaczenia. Ten kod:

```
int a = 5;
a++;
printf("%d\n", a);
```

da taki sam wynik jak ten:

```
int a = 5;
++a;
printf("%d\n", a);
```

Należy zwrócić uwagę, że operatory inkrementacji i dekrementacji modyfikują wartość, przy której zostały umieszczone. Modyfikacja wartości następuje też oczywiście w przypadku operatora przypisania, modyfikowana jest wartość zmiennej znajdującej się po jego lewej stronie. Pozostałe operatory arytmetyczne zwracają wartość, którą można przypisać do zmiennej lub przekazać do funkcji:

```
int a, b, c;
a = 2;
b = 3;
c = a + b;
printf("a=%d b=%d c=%d a+b=%d\n",
a, b, c, a+b);
```

Jeśli chcemy zwrócić wynik działania operatora do zmiennej, która jest z jego lewej strony, możemy wykorzystać zapis skrótowy, w którym dołączamy operator przypisania. Przykładowo zamiast:

```
a = a + 3;
```

możemy napisać:

```
a += 3;
```

Wśród operatorów arytmetycznych nie ma operatora potęgowania, znanego np. z Basica. Potęgowanie, pierwiastkowanie i inne operacje matematyczne można wykonywać za pomocą funkcji zawartych w bibliotece math.h.

Tabela 4 pokazuje **operatory porównania**, a **tabela 5** – **operatory logiczne**, które jako wynik swojego działania zwracają wartość logiczną: prawdę lub fałsz, odwzorowaną odpowiednio jako 1 lub 0. Natomiast operand traktowany jest jako prawda tak długo, jak długo jest różny od zera. Działanie operatorów logicznych zostanie bardziej szczegółowo przedstawione przy okazji omawiania warunków oraz pętli.

Tabela 6 pokazuje **operatory bitowe**. Operator NOT ustawia wartości wszystkich bitów w zmiennej na przeciwne. Podobnie operacje AND, OR i XOR działają na parach odpowiadających sobie bitów z poszczególnych zmiennych. Operacji AND i OR używamy często w stosunku do masek bitowych. Np. aby wyzerować dwa najmłodsze bity w zmiennej typu char, napiszemy:

```
a &= 0xfc;
```

aby ustawić najstarszy bit, napiszemy:

```
a |= 0x80;
```

Przesunięcie bitowe przesuwa bity w zmiennej a o b miejsc. Jest często wykorzystywane zamiast mnożenia razy potęga liczby 2, ponieważ na wielu procesorach jest szybsze od standardowej operacji mnożenia. Np. przesuwając bity o cztery miejsca w lewo, wykonujemy mnożenie razy 2^4, czyli razy 16. Analogicznie przesuwanie w prawo powoduje dzielenie przez potęgę dwójki. Bity, które „wypadają" ze zmiennej, są tracone, „puste" miejsca z drugiej strony są uzupełniane zerami. Nie zachodzi więc rotacja bitów w zmiennej, zachowany natomiast zostaje bit znaku.

Wartość zmiennej podczas testowania operatorów bitowych najlepiej obserwować w debuggerze, przy włączonym widoku szesnastkowym (Hexadecimal Display w menu podręcznym okienka Locals). Funkcja printf wykonuje bowiem konwersję na int i zmienną char o wartości 0xff wyświetli jako ffffffff.

Operatorów możemy używać oczywiście nie tylko na zmiennych, ale też na wartościach zwracanych przez funkcje, np. wyrażenie:

```
fun1() > fun2()
```

zwróci 1, jeśli wartość zwrócona przez funkcję fun1 będzie większa od wartości zwróconej przez fun2. Wydawać się to może proste, jednak jest pewien haczyk. Otóż w przypadku operatorów logicznych nie zawsze jest konieczne sprawdzanie wartości obu operandów, aby wynik był znany. Jeśli wykonujemy sumę logiczną (alternatywę, OR), to wówczas jeśli pierwszy operand wynosi 1, to wynik też wyniesie 1, niezależnie od tego czy drugi operand będzie zerem czy jedynką. Analogicznie, jeśli wykonujemy iloczyn logiczny (koniunkcję, AND), a pierwszy operand jest zerem, to wynik też będzie zerem, obojętnie, jaką wartość będzie miał drugi operand. W języku C przyjęto, że jeśli operand nie ma znaczenia, to jego wartość nie jest obliczana. Oznacza to, że jeśli operandem jest wywołanie funkcji, to nie zosta-

Tabela 3

Operator	Składnia
przypisanie	a = b
dodawanie	a + b
odejmowanie	a - b
mnożenie	a * b
dzielenie	a / b
modulo	a % b
inkrementacja	++a i a++
dekrementacja	--a i a--

Tabela 4

Operator	Składnia
równe	a == b
nierówne	a != b
większe	a > b
mniejsze	a < b
większe lub równe	a >= b
mniejsze lub równe	a <= b

Tabela 5

Operator	Składnia
negacja logiczna	!a
koniunkcja logiczna	a && b
alternatywa logiczna	a || b

Tabela 6

Operator	Składnia
NOT	~a
AND	a & b
OR	a | b
XOR	a ^ b
przesunięcie w lewo	a << b
przesunięcie w prawo	a >> b

```
#include <stdio.h>

int printHello();
int printWorld();

int main(void) {
    if (printHello() && printWorld()) {
        printf("!");
    }
    return 0;
}

int printHello() {
    printf("Hello");
    return 1;
}

int printWorld() {
    printf("World");
    return 1;
}
```

nie ona wywołana. Popatrzmy na przykład w ramce na sąsiedniej stronie.

Zostanie wyświetlony napis „Hello World!”. Jeśli jednak zmodyfikujemy funkcję printHello, aby zwracała zero, pojawi się tylko „Hello”.

Napis „Hello World” pojawi się, jeśli printHello będzie zwracać wartość niezerową. Cały napis, razem z wykrzyknikiem, będzie mógł być wyświetlony, jeśli obie funkcje zwrócą wartości niezerowe.

W **tabeli 7** podana jest **kolejność operatorów** (wraz z łącznością). Podobnie jak w matematyce, w języku C obowiązuje kolejność operatorów, np. mnożenie i dzielenie wykonywane jest przed dodawaniem i odejmowaniem. Nie trzeba koniecznie znać tabeli na pamięć, ale warto się z nią pobieżnie zaznajomić, szczególnie analizując kod pisany przez inną osobę. Pisząc własne programy, w razie wątpliwości zwykle po prostu dodaje się nawiasy. Daje to pewność, że operatory zostaną użyte w przewidywanej przez nas kolejności.

Kolejność	Operator	Opis	Łączność
1	++ --	postinkrementacja, postdekremenacja	lewostronna
	()	wywołanie funkcji	
	[]	indeksowanie tablicy	
	.	dostęp do elementu struktury lub unii	
	->	dostęp do elementu struktury lub unii przez wskaźnik	
	(typ){lista}	literał złożony (standard C99)	
2	++ --	preinkrementacja i predekrementacja	prawostronna
	+ -	znak wartości liczbowej (dodatnia lub ujemna)	
	! ~	zaprzeczenie logiczne, zaprzeczenie bitowe	
	(typ)	rzutowanie	
	*	wyłuskanie (dereferencja)	
	&	adres	
	sizeof	rozmiar	
	_Alignof	wartość wyrównania (standard C11)	
3	* / %	mnożenie, dzielenie, reszta z dzielenia (modulo)	lewostronna
4	+ -	dodawanie i odejmowanie	
5	<< >>	przesunięcie bitowe w lewo, w prawo	
6	< <=	mniejsze, mniejsze lub równe	
	> >=	większe, większe lub równe	
7	== !=	równe, różne	
8	&	bitowe AND	
9	^	bitowe XOR	
10	|	bitowe O	
11	&&	logiczne AND	
12	||	logiczne OR	
13	?:	wyrażenie warunkowe	prawostronna
14	=	przypisanie zwykłe	
	+= -=	przypisanie z sumowaniem, z odejmowaniem	
	*= /= %=	przypisanie z mnożeniem, z dzieleniem, z modulo	
	<<= >>=	przypisanie z przesunięcie w lewo, w prawo	
	&= ^= |=	przypisanie z bitowym AND, XOR, OR	
15	,	separacja argumentów lub elementów	lewostronna

Tabela 7

Warunki

Ważnymi konstrukcjami języka programowania są pętle i warunki. Są to bloki kodu, których wykonanie zależy od wartości zmiennych. Proste warunkowe wykonanie kodu realizowane jest za pomocą słowa kluczowego if i ewentualnie else. Ogólna składnia to:

```
if (warunek) {
        operacje1;
} else {
        operacje2;
}
```

Jako warunek możemy umieścić praktycznie wszystko, co zwraca wartość. Może to być porównanie zmiennych, wywołanie funkcji czy też pojedyncza zmienna. Wykonana zostanie operacja1, jeśli sprawdzany warunek będzie prawdziwy czyli będzie wartością różną od zera. Jeśli warunek nie będzie spełniony, zostanie wykonana operacja2. Operacje1 i operacje2 symbolizują tu jedną lub więcej operacji wykonywanych w naszym programie. Jeśli operacja jest tylko jedna, możemy opuścić nawiasy klamrowe. Jeśli wykonujemy kod tylko dla spełnienia warunku, możemy opuścić słowo kluczowe else. Przykładowo, jeśli chcemy zwiększyć zmienną b o 2 pod warunkiem, że zmienne a i b są równe, możemy napisać:

```
if (a == b) b += 2;
```

Jeśli będziemy chcieli wypisać komunikaty w zależności od tego, czy zmienna a jest zerem, możemy napisać:

```
if (a) printf("Zmienna a nie jest zerem."); else printf("Zmienna a jest zerem");
```

Przy bardziej złożonym kodzie możemy rozbić zapis na więcej linijek, zastosować wcięcia i nawiasy, jeśli operacji jest więcej:

```
if (a > b)
        printf("a jest wieksze od b.\n");
else {
        printf("a jest mniejsze lub rowne b.\n");
        c++;
}
```

Uwaga: choć mamy XXI wiek, windowsowa konsola nadal używa w polskich systemach DOSowej strony kodowej 852. Tymczasem edytor kodu źródłowego stosuje windowsową stronę kodową 1250. Polskie litery wyświetlą się poprawnie, jeśli program zostanie uruchomiony z wiersza polecenia (nie ze środowiska Visual C++), a uprzednio zostanie wykonana komenda:

```
chcp 1250
```

Musi być też ustawiona odpowiednia czcionka dla okienka wiersza polecenia, np. Lucida Console. Podczas nauki języka C, dla uproszczenia, można zrezygnować z polskich liter. Tak też zostało zrobione w tym przykładzie.

Jeśli chcemy sprawdzić kilka wartości dla jednego warunku, możemy użyć konstrukcji switch. Ma ona postać:

```
switch (wyrażenie)
{
    case wartość1:
       operacje1;
    case wartość2:
       operacje2;
       break;
    default:
       operacje3;
}
```

Po obliczeniu wartości wyrażenia jest ona porównywana z kolejnymi wartościami przy słowach kluczowych case. Jeśli są równe, wykonywane są odpowiednie operacje. Ważne jest tu słowo kluczowe break. Jeśli zostanie dopasowana dana wartość, program wykonuje skok do danych operacji i wykonywane są kolejne instrukcje z bloku switch tak długo, jak długo nie zostanie napotkane słowo kluczowe break. W powyższym przykładzie, jeśli wyrażenie będzie miało taką samą wartość jak wartość1, zostaną wykonane operacje oznaczone jako operacje1 oraz operacje2, pomimo że wartość wyrażenia będzie różna od wartość2. Słowo kluczowe default oznacza kod, który zostanie wykonany, gdy nie została dopasowana żadna wartość. Należy pamiętać, że wartości przy słowie case muszą być znanymi podczas kompilacji liczbami, nie mogą być zmiennymi ani wywołaniami funkcji. Przykładowo:

```
switch (a)
{
    case 2:
       printf("Zmienna a ma wartosc 2.\n");
       break;
    case 7:
       printf("Zmienna a ma wartosc 7.\n");
       break;
    default:
       printf("Zmienna a nie ma wartosci 2 ani 7.\n");
}
```

C oferuje także specjalną konstrukcję do szybkiego zwracania wartości na podstawie warunku. Wygląda ona następująco:

```
warunek ? wartość_dla_prawdy : wartość_dla_fałszu
```

Poniżej użyjemy jej do wyznaczenia większej z dwóch liczb:

```
#include <stdio.h>

int main(void) {
    int x = 1;
    int y = 2;
    printf("max(%d, %d) = %d\n", x, y, x > y ? x : y);
    return 0;
}
```

Język C nie ma dedykowanych typów boolowskich, przeznaczonych do przechowywania wartości typu prawda oraz fałsz. Zamiast nich najczęściej wykorzystuje się typ int. Czasami spotyka się definicje podobne do poniższych:

```
#define TRUE 1
#define FALSE 0
```
Można dzięki temu zastosować zapis:
```
x = FALSE;
```
czy też:
```
if (a == TRUE) {
        ...
}
```

Używając takich makr w warunkach, trzeba pamiętać, że w C za prawdę uznaje się wartości i wyrażenia różne od zera. Jeśli w powyższym przypadku zmienna a będzie miała wartość np. 5, warunek nie zostanie spełniony. Konstrukcji takiej można użyć wtedy, gdy mamy pewność, że dana zmienna będzie przyjmować tylko wartości 0 i 1.

Programista przed rozpoczęciem pisania kodu musi zastanowić się, co chce zrealizować i jak to najlepiej zapisać. Ponieważ język C pozwala na zapisanie jednej rzeczy na wiele sposobów, warto wybrać taką, która jest najbardziej czytelna. Jeśli nawet nie pracujemy w zespole i nasz kod nie jest czytany przez inne zespoły, zapewne będziemy do niego wracać, np. rozbudowując go lub wprowadzając poprawki. Czytelność warunków jest szczególnie ważna, gdyż pozwala zorientować się w logice programu. W C możemy warunek zapisać w krótkiej formie:
```
if (a) { }
```
lub w dłuższej:
```
if (a != 0) { }
```
Obie są poprawne i obie mają swoje wady i zalety. Forma pierwsza jest krótsza, zajmuje mniej miejsca i może się wydawać zgrabniejsza. Z drugiej strony wymaga pamiętania, że następuje sprawdzenie, czy zmienna jest różna od zera. W przypadku drugiej formy jest to zapisane wprost i od razu widać, co autor miał na myśli. Pierwsza forma jest odpowiednia, gdy zmienna przyjmuje wartości 0 lub 1 i ogólnie traktowana jest jako boolowska. Jeśli zmienna może przyjmować różne wartości, bardziej odpowiednie będzie przyrównanie do zera.

Znaczenie ma też odpowiednie nazewnictwo zmiennych. W niniejszym kursie używamy bardzo krótkich i prostych przykładów, w których nazwy zmiennych nie mają większego znaczenia. Pisząc jednak właściwy program, trzeba zadbać o w miarę krótkie, ale opisowe nazwy. To samo dotyczy funkcji. Dzięki temu kod będzie czytelny i samodokumentujący się. Jak wspomniano, warunki można zapisywać w skróconej formie, jeśli zmienna przyjmuje tylko dwie wartości. Dotyczy to np. zmiennych przechowujących wynik jakiejś akcji, np. sprawdzenia czy plik istnieje. Można wtedy zapisać:
```
if (fileExists) { }
```
Wygląda to zgrabniej, niż gdyby dodać sprawdzenie, czy zmienna jest różna od zera. Natomiast jeśli mamy np. zmienną, która zlicza impulsy (przyjmuje wartości nieujemne) i chcemy sprawdzić, czy jest większa od zera, lepiej zapisać to wprost:
```
if (pulsesCounter > 0) { }
```
niż skracać do:
```
if (pulsesCounter) { }
```
choć druga forma też byłaby poprawna.

Czasami można spotkać wyrażenie **Ramka 3**
```
!!zmienna
```
które może się wydawać bez sensu, gdyż mamy do czynienia z podwójnym zaprzeczeniem, a więc zwrócona wartość logiczna będzie taka, jakby wykrzykników w ogóle nie było. Po co zatem stosować taką konstrukcję? Otóż dzięki niej zwrócona będzie wartość wynosząca zawsze tylko 0 lub 1, nawet jeśli w zmiennej będzie jakaś duża liczba. Czasami takie ograniczenie zakresu wartości może być potrzebne. Zapis z podwójnym wykrzyknikiem jest równoważny:
```
zmienna ? 1 : 0
```
ale jest krótszy. Opisana konstrukcja dotyczy nie tylko zmiennych, może być też stosowana do wywołań funkcji lub innych wyrażeń.

Pętle

Przechodzimy teraz do pętli, bloków kodu, które mogą być wykonywane wielokrotnie. Najprostsza pętla to pętla typu while:
```
while (warunek) {
        operacje;
}
```
Instrukcje zapisane tutaj jako operacje powtarzane są tak długo, jak długo warunek jest spełniony. Popatrzmy na przykład:
```
int i = 10;
while (i) {
        i--;
        printf("%d\n", i);
}
```
W pętli sprawdzany jest warunek, czy zmienna i ma wartość różną od zera. Początkowo jest to prawdą, jednak zmienna zainicjalizowana wartością 10 jest za każdą iteracją pętli dekrementowana i osiąga w końcu wartość 0. W ten sposób zrealizowaliśmy pętlę wykonującą się 10 razy. Zapis możemy skrócić przenosząc dekrementację do warunku:
```
int i = 10;
while (i--) printf("%d\n", i);
```

Pętla while ma jeszcze swoją drugą formę, wykorzystującą słowo kluczowe do:
```
do {
        operacje;
} while (warunek);
```
Zapis podpowiada, jak działa taka pętla: operacje wykonywane są co najmniej raz, ponieważ warunek sprawdzany jest dopiero po ich wykonaniu, a nie przed. Pierwsza postać pętli while jest częściej spotykana, ale forma z do też czasem bywa przydatna. Wszystko zależy od algorytmu jaki realizować ma nasz program.

Język C ma też pętlę for, o następującej postaci:
```
for (inicjalizator; warunek; aktualizacja) {
      operacje;
}
```
Pierwsze z wyrażeń sterujących pętlą jest wykonywane tylko raz, przed pierwszą iteracją pętli. Zazwyczaj umieszcza się tam inicjalizację zmiennej pełniącej funkcję licznika. Drugie wyrażenie to warunek sprawdzany przed każdą iteracją. Jeśli warunek nie będzie spełniony, pętla zostanie przerwana i program przejdzie do dalszych instrukcji. Ostatnie wyrażenie wykonywane jest po każdej iteracji. Zazwyczaj umieszczana jest tam dekrementacja lub inkrementacja zmiennej licznikowej. Pętla wykonująca się 10 razy może zostać zapisana następująco:
```
for (int i = 0; i < 10; i++)
        printf("%d\n", i);
```
W ramach inicjalizacji deklarowana jest zmienna licznikowa i inicjalizowana jest ona wartością 0. Warunek to wartość zmiennej mniejsza niż 10. W ten sposób pętla wykona się dla wartości od 0 do 9. Aktualizacja zmiennej i następuje przez jej zwiększanie o 1. W ciele pętli wyświetlana jest wartość zmiennej licznikowej.

Każde z trzech wyrażeń w pętli for można opuścić, średniki jednak trzeba pozostawić. Jeśli usuniemy warunek a wnętrze pętli będzie pozbawione instrukcji pozwalających na wyjście z niej, pętla będzie się wykonywać w nieskończoność. Pętla for(;;) odpowiada pętli while(1).

Wykonaniem pętli sterują nie tylko sprawdzane przez nie warunki. Programista ma także do dyspozycji słowa kluczowe break oraz continue. Break powoduje natychmiastowe wyjście z pętli. Natomiast continue przerywa wykonywanie bieżącej iteracji i rozpoczyna następną. Popatrzmy na przykłady:
```
for (int i = 0; i < 10; i++) {
     if (i==8) break;
     if (i==5) continue;
     printf("%d\n", i);
}
```
W wyniku działania tej pętli wyświetlone zostaną liczby 0, 1, 2, 3, 4, 6, 7. Słowo kluczowe continue spowoduje opuszczenie liczby 5, a break spowoduje wyjście z pętli po osiągnięciu wartości 8, ale przed jej wypisaniem.

Grzegorz Niemirowski
grzegorz@grzegorz.net