Jarosław **Stańczyk**

NOWOCZESNY C

Przegląd C23 z przykładami

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/nowocp
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-9995-2

Printed in Poland.

Spis treści

III. Biblioteka standardowa

9. Instrukcje

Instrukcja określa czynność do wykonania, na przykład operację arytmetyczną lub wywołanie funkcji. Wiele instrukcji służy do sterowania przebiegiem programu poprzez definiowanie pętli i rozgałęzień. Instrukcje są przetwarzane kolejno, z wyjątkiem sytuacji, gdy instrukcje sterujące powodują skoki.

9.1. Instrukcja prosta

Każda instrukcja, która nie jest blokiem, jest zakończona średnikiem.

Składnia:

instrukcja$^{(opcja)}$;
lub
lista-atrybutów instrukcja;

Instrukcja składająca się tylko ze średnika nazywana jest *instrukcją pustą* i nie wykonuje żadnej operacji. Opcjonalnie instrukcja może być poprzedzona *listą-atrybutów* (patrz rozdział 6.). Uwaga, *lista-atrybutów* zakończona średnikiem tworzy deklarację atrybutu.

Przykład 9.1. Przykłady instrukcji prostych (fragment programu prg09-01.c)

```
1 y = x;                        // instrukcja przypisania
2 printf("%d, %s\n", y, str); // wywołanie funkcji
3
4 for ( int i = 0; str[i] ; ++i )
5 {
6     ;   // instrukcja pusta
7 }
```

9.2. Bloki

Blok, nazywany również *instrukcją złożoną*, grupuje kilka instrukcji w jedną. Blok może zawierać deklaracje. Deklaracje w bloku zwykle poprzedzają instrukcje. Norma zezwala jednak na umieszczanie deklaracji w dowolnym miejscu.

Składnia:

```
{
    lista deklaracji(opcja)
    lista instrukcji(opcja)
}
```

W dowolnym miejscu bloku może być definiowany nowy blok. Zwykle blok jest tworzony wszędzie tam, gdzie składnia wymaga instrukcji, natomiast program wymaga ich kilku. Dzieje się tak na przykład, gdy więcej niż jedna instrukcja ma zostać powtórzona w pętli. Blok rozpoczyna się i kończy nawiasem klamrowym (bez średnika na końcu). Instrukcje wewnątrz bloku zakończone są średnikiem.

Przykład 9.2. Instrukcje blokowe (fragment programu prg09-02.c)

```
{   // przykład bloku

    // deklaracje
    int i = 0;
    static long a;
    extern long max;

    // instrukcje
    ++a;
    if (a >= max)
    {
        // blok zagnieżdżony
        ...
    }
    ...
}
...
{
    // pusty blok
}
```

9.3. Instrukcje sterujące

Instrukcje sterujące służą do sterowania przebiegiem programu, zalicza się do nich:

— instrukcje warunkowe:

`if ... else`

`switch`

— pętle:

`while`

`do ... while`

`for`

— skoki:

`goto`

`continue`

`break`

`return`

Przeanalizujmy poszczególne z nich.

9.3.1. Instrukcje warunkowe

Instrukcja `if`

Instrukcja `if` powoduje *skok warunkowy*.

Składnia:

lista-atrybutów$^{(opcja)}$ `if (` *wyrażenie* `)` *instrukcja*
lub
lista-atrybutów$^{(opcja)}$ `if (` *wyrażenie* `)` *instrukcja* `else` *instrukcja2*

Pierwsze oceniane jest *wyrażenie*, które musi być typu skalarnego. Jeśli *wyrażenie* jest prawdziwe (tzn. wynik wyrażenia jest niezerowy), wykonywana jest *instrukcja*. W przeciwnym razie, jeśli występuje `else`, wykonywana jest *instrukcja2*.

Użycie `else` jest opcjonalne. Jeśli wartość *wyrażenia* wynosi `0` (*fałsz*), a `else` nie występuje, wówczas realizacja programu jest kontynuowana od następnej instrukcji.

Jeśli zagnieżdżonych jest kilka instrukcji `if`, klauzula `else` zawsze należy do *ostatniej* klauzuli `if`, która nie ma jeszcze klauzuli `else`. `else` można przypisać do innego `if`, tworząc jawne bloki.

Przykład 9.3. Instrukcja `if` (fragment programu `prg09-03.c`)

```
 1 if (x > y)
 2     max = x;     // Przypisz większe z x i y do
 3 else             // zmiennej max.
 4     max = y;
 5
 6 if (n > 0)
 7 {
 8     if (n % 2 == 0)
 9         puts ("n parzyste dodatnie");
10 }
11 else             // else należy do pierwszego if
12     puts ("n jest mniejsze od zera");
```

Pewne warunki można uprościć, na przykład zwykle stosuje się:

```
if ( wyrażenie )
```

zamiast

```
if ( wyrażenie != 0 )
```

Patrz operator warunkowy `:?` opisany w rozdziale 8.

Instrukcja `switch`

W instrukcji `switch` wartość *wyrażenia* jest porównywana ze stałymi skojarzonymi z etykietami przypadków `case`. Jeśli wynikiem wyrażenia jest stała skojarzona z etykietą przypadku, wykonywanie programu jest kontynuowane od pasującej etykiety. Jeśli nie ma pasującej etykiety, wykonanie programu przechodzi do etykiety domyślnej (`default`), jeśli jest zadeklarowana, w przeciwnym razie wykonanie jest kontynuowane od instrukcji występującej po instrukcji `switch`.

Składnia:

lista-atrybutów$^{(opcja)}$ `switch ( ` *wyrażenie* ` )` *instrukcja-switch*

Wyrażenie jest wyrażeniem całkowitoliczbowym, a *instrukcja-switch* jest instrukcją blokową z etykietami `case` i co najwyżej jedną etykietą domyślną `default`.

Każda etykieta `case` ma postać:

lista-atrybutów$^{(opcja)}$ `case` *wyrażenie-stałe*`:` *instrukcja-case*

gdzie *wyrażenie-stałe* jest stałym wyrażeniem całkowitoliczbowym. Wszystkie stałe muszą się od siebie różnić.

Etykieta `default`, jeśli występuje, ma postać:

lista-atrybutów$^{(opcja)}$ `default:` *instrukcja-default*

Po skoku od `switch` do `case` wykonywanie programu jest kontynuowane sekwencyjnie, niezależnie od innych etykiet.

Instrukcja `break` może zostać użyta do wyjścia z bloku `switch` w dowolnym momencie. `break` jest konieczna, jeśli instrukcje następujące po innych etykietach `case` nie mają być wykonywane.

Przykład 9.4. Instrukcja `switch` (fragment programu `prg09-04.c`)

```
 1 switch (polecenie)  // w zależności od polecenia, np. od wartości
 2 {                   // wybranej przez użytkownika w menu
 3     case 'a':
 4     case 'A':       // jeśli wynikiem jest 'a' lub 'A'
 5         akcja1 ();  // wykonaj akcja1(),
 6         break;      // a następnie opuść blok switch
 7
 8     case 'b':
 9     case 'B':       // jeśli wynikiem jest 'b' lub 'B'
10         akcja2 ();  // wykonaj akcja2()
11         break;      // i opuść blok
12
13     default:        // dla każdego innego wyniku polecenia:
14         putchar ('\a');  // wykonaj tę instrukcję
15 }
```

9.3.2. Pętle

Pętla składa się z instrukcji lub bloku zwanego *treścią pętli*, który jest wykonywany wielokrotnie, w zależności od warunku. C oferuje trzy instrukcje do tworzenia pętli:

— `while`

— `do...while`

— `for`

W każdej z tych pętli liczba wykonanych iteracji treści pętli jest określana przez *instrukcję sterującą*. Jest to wyrażenie typu skalarnego.

Instrukcja `while`

Składnia:

```
    lista-atrybutów(opcja) while ( wyrażenie ) instrukcja
```

Instrukcja `while` jest pętlą *sterowaną od góry*: tzn. jako pierwszy sprawdzany jest warunek pętli (czyli *wyrażenie*). Jeśli zwróci wartość *prawda*, wykonywana jest treść pętli (czyli *instrukcja*), a następnie ponownie oceniany jest warunek pętli. Jeśli warunek pętli jest *fałszywy*, wykonywanie programu jest kontynuowane od pierwszej instrukcji występującej po treści pętli.

Przykład 9.5. Pętla `while` (fragment programu `prg09-05.c`)

```
1 s = str;            // niech wskaźnik s
2 while( *s != '\0')  // wskazuje koniec łańcucha str
3     ++s;
```

Instrukcja `do ... while`

Składnia:

```
lista-atrybutów(opcja) do instrukcja while ( wyrażenie ) ;
```

Instrukcja `do ... while` jest pętlą *sterowaną od dołu*: najpierw wykonywana jest treść pętli, a następnie oceniane jest wyrażenie sterujące. Jest to powtarzane, dopóki wyrażenie sterujące nie będzie miało wartości *fałsz*.

Kluczową różnicą w stosunku do instrukcji `while` jest to, że treść pętli `do ... while` jest zawsze wykonywana przynajmniej raz. A pętla `while` może w ogóle nie zostać wykonana, ponieważ jej wyrażenie może już na starcie mieć wartość *false*.

Przykład 9.6. Pętla `do-while` (fragment programu `prg09-06.c`)

```
1 i = 0;
2 do                          // kopiuj łańcuch str1
3     str2[i] = str1[i];      // do str2
4 while ( str1[i++] != '\0' );
```

Instrukcja `for`

Składnia:

```
lista-atrybutów(opcja)
for ( wyrażenie1(opcja) ; wyrażenie2(opcja) ; wyrażenie3(opcja) ) instrukcja
```

Typowa pętla `for` używa *wyrażenia2* jako sterującego i jest ono oceniane przed każdym wykonaniem pętli. Wyrażenie to musi być typu skalarnego. *wyrażenie3* jest wykonywane po realizacji każdej pętli. *wyrażenie1* inicjalizuje pętlę, jest wykonywane raz, przed rozpoczęciem pierwszej pętli, jeszcze przed oceną *wyrażenia2*. Jeśli *wyrażenie1* jest deklaracją, jej zakres ograniczony jest do pętli `for`. *wyrażenie1* i *wyrażenie3* mogą być dowolnymi wyrażeniami, można je pominąć. Pominięte *wyrażenie2* jest zastępowane przez niezerową stałą.

Pętla:

```
for ( wyrażenie1; wyrażenie2; wyrażenie3 ) instrukcja
```

jest równoważna pętli:

```
wyrażenie1
while ( wyrażenie2 )
{
    instrukcja
    wyrażenie3
}
```

Przykład 9.7. Pętla `for` i jej równoważnik `while` (fragment programu `prg09-07.c`)

```
1 for (int i = DELAY; i > 0; --i)
2     ;                 // poczekaj chwilę
3
4 // równoważna pętla while:
5 int i = DELAY;        // inicjalizacja
6 while ( i > 0)        // test wyrażenia sterującego
7     --i;              // aktualizacja zmiennej i
```

Uwaga, powyższe pętle różnią się zakresem zmiennej `i`.

9.3.3. Skoki bezwarunkowe

Instrukcje skoku powodują bezwarunkowy skok do innego miejsca w kodzie.

Instrukcja `goto`

Instrukcja `goto` przeskakuje do dowolnego miejsca w funkcji, w której jest wywołana. Miejsce docelowe skoku określa nazwa etykiety.

Składnia:

```
lista-atrybutów(opcja) goto etykieta;
```

etykieta to nazwa, po której następuje dwukropek pojawiający się przed instrukcją, do której ma nastąpić skok. Ograniczeniem instrukcji `goto` jest to, że zarówno `goto`, jak i *etykieta* muszą być zawarte w tej samej funkcji.

Składnia:

```
lista-atrybutów(opcja) etykieta:
```

Przykład 9.8. Typowy skok przy użyciu `goto`

```
1 for ( ... )             // wyskok z
2     for ( ... )         // zagnieżdżonych pętli
3         if ( error )
4             goto obsluga_bledu;
5 ...
6 obsluga_bledu:          // tu zaczyna się obsługa błędów
7 ...
```

Biblioteka standardowa C dostarcza funkcje umożliwiające *skoki odległe*. Więcej o skokach odległych w rozdziale 24.

Instrukcja `continue`

Instrukcja `continue` może być używana tylko wewnątrz pętli. Powoduje ominięcie pozostałej części ciała pętli (bloku). Tak więc sterowanie wewnątrz `while` lub `do ... while` po natrafieniu na `continue` przechodzi do następnego testu *wyrażenia* sterującego, a w pętli `for` przeskakuje do oceny *wyrażenia2*.

Składnia:

```
lista-atrybutów(opcja) continue;
```

Przykład 9.9. Instrukcja `continue`

```
for (i = -10; i < 10; ++i)
{
    ...
    if (i == 0)
        continue;   // dla wartości 0 pomiń resztę ciała pętli
    ...
}
```

Instrukcja `break`

Instrukcja `break` wyskakuje z pętli do następnej instrukcji za ciałem pętli lub instrukcji `switch`.

Składnia:

```
lista-atrybutów(opcja) break;
```

Przykład 9.10. Instrukcja `break`

```
1 while (1)
2 {
3     ...
4     if (polecenie == ESC)
5         break;   // jeśli polecenie ESC, zakończ pętlę
6     ...
7 }
```

Instrukcja `return`

Instrukcja `return` kończy wykonywanie bieżącej funkcji i zwraca kontrolę do obiektu wywołującego. W funkcji może pojawić się dowolna liczba instrukcji `return`. Wartość wyrażenia w instrukcji `return` jest zwracana wywołującemu jako wartość funkcji, w razie potrzeby wartość ta jest konwertowana na odpowiedni typ funkcji.

Składnia:

```
lista-atrybutów(opcja) return wyrażenie(opcja) ;
```

Wyrażenie w instrukcji `return` można pominąć. Ma to jednak sens tylko w funkcjach typu `void`, w takim przypadku całą instrukcję `return` można również pominąć. Wówczas funkcja zwraca kontrolę do obiektu wywołującego po zakończeniu swego bloku funkcyjnego.

Przykład 9.11. Intrukcja `return` (fragment programu `prg09-11.c`)

```
1 int max (int a, int b)  // maksimum z a i b
2 {
3     return (a > b ? a : b);
4 }
```

Indeks

ELEMENTARZ JĘZYKA C

Mimo że C — stworzony przez pracownika Laboratoriów Bella Dennisa Ritchiego — liczy sobie pół wieku, wciąż pozostaje niezwykle popularnym proceduralnym językiem ogólnego zastosowania. To dzięki niemu powstają elementy systemów operacyjnych i programów użytkowych. Tymczasem za sprawą Międzynarodowej Organizacji Normalizacyjnej (ISO) światło dzienne ujrzy najnowsza wersja standardu języka, która ukaże się pod koniec 2023 roku.

Właśnie tej iteracji poświęcony jest podręcznik Jarosława Stańczyka *Nowoczesny C. Przegląd C23 z przykładami*. Treść została podzielona na trzy części. W pierwszej autor opisuje budowę programu i kod źródłowy języka C, a także przedstawia narzędzia do kompilacji i uruchamiania przykładów dołączonych do książki. Jej trzon stanowią pozostałe dwie części, które zawierają między innymi omówienie języka C i standardowej biblioteki. Prezentowanym zagadnieniom towarzyszą liczne przykłady — przy każdym podano nazwę pliku źródłowego. Wszystkie kody źródłowe można samodzielnie pobrać z serwisu GitHub.

W książce między innymi:

- **nowości i zmiany wprowadzone w C23**
- **elementy leksykalne**
- **podstawowe typy danych**
- **stałe, wyrażenia i operatory**
- **instrukcje, funkcje**
- **dyrektywy preprocesora**
- **biblioteka standardowa**

Z językiem C za pan brat!

Jarosław Stańczyk — absolwent Politechniki Wrocławskiej, doktor nauk technicznych w dziedzinie informatyki. Pracował na Uniwersytecie w Magdeburgu jako stypendysta Niemieckiej Fundacji ds. Badań (Deutsche Forschungsgemeinschaft). Zajmuje się tworzeniem oprogramowania systemów wbudowanych. Pasjonat informatyki i robotyki, szczególnie zainteresowany emocjami maszyn i relacją człowiek – maszyna. Entuzjasta wolnego oprogramowania.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-9995-2

9 788328 399952

Cena: 119,00 zł