Dyrektywy

Dyrektywy preprocesora to specjalne polecenia stosowane w kodzie programu, które powodują odpowiednią kompilację programu, jednak nie wpływają na sam przebieg programu po jego skompilowaniu.

Preprocesor jest programem, dokonującym wstępnej obróbki kodu źródłowego przed jego kompilacją.

#include

Dyrektywa ta powoduje, że kompilator w miejscu jej wystąpienia wstawia wskazany plik z kodem źródłowym. Ogólne, równoważne postacie:

#include "*nazwa_pliku*"
#include <*nazwa_pliku*>

Możliwe jest stosowanie samej nazwy pliku (bez ścieżki dostępu) znajdującego się w katalogu pliku, z którego dyrektywa jest wywołana, w katalogu plików nagłówkowych ustawionych w kompilatorze, lub też pełna nazwa ze ścieżką pliku.

Dyrektywa
#include<iostream>
Informuje kompilator, aby pobrał plik „iostream”,
który zawiera deklaracje strumieni wejścia i wyjścia: cin, cout,
operatora wyjścia(<<) i operatora ekstrakcji (>>)
Umożliwia to wykonanie poprawnego łączenia kodu wynikowego
biblioteki „iostream” z instrukcjami wejscia/wyjscia w programie.

Operatory arytmetyczne

Operator	Działanie	Przykład
+	dodawanie	a = b + c;
-	odejmowanie	a = b - c;
*	mnożenie	a = b * c;
/	dzielenie	a = b / c;
%	reszta z dzielenia (modulo)	a = b % c;

Uwagi

Wszystkie operatory za wyjątkiem operatora % można stosować zarówno do argumentów *całkowitych*, jak i *zmiennoprzecinkowych*. Operatory + i - można również stosować jako operatory *jednoargumentowe*.

Jeśli przy dzieleniu liczb całkowitych iloraz zawiera część ułamkową, jest ona odrzucana.

Przykłady:

25 / 7	→	3;
25 / 7.	→	3.571428
35. / 5	→	7.0
1 / 4	→	0
19 % 6	→	1
0 % 5	→	0
18 % 6	→	0

Operatory relacji

Operator	Działanie	Przykład
<	mniejszy	a < b
<=	mniejszy lub równy	a <= b
>	większy	a > b
>=	większy lub równy	a >= b
==	równy	a ==b
!=	nie równy	a != b

Operatory logiczne

Operator	Działanie	Przykład
!	negacja	! a
&&	koniunkcja (iloczyn logiczny)	a && b
||	alternatywa (suma logiczna)	a || b

Operatory logiczne

Wyrażenia połączone dwuargumentowymi operatorami logicznymi koniunkcji i alternatywy zawsze *są wartościowane od strony lewej do prawej*. Kompilator oblicza wartość wyrażenia dotąd, dopóki na pewno nie wie jaki będzie wynik.

Oznacza to, że w wyrażeniu

```
( a == 0 ) && ( m == 5 ) && ( x > 23 )
```

kompilator będzie obliczał od lewej do prawej, a jeśli pierwszy czynnik koniunkcji nie będzie prawdziwy, dalsze obliczanie zostanie przerwane.

Operator sizeof

- Operator sizeof pozwala nam rozpoznać zachowania kompilatora i komputera, z którymi przyszło nam pracować. Jest to ważne z dwóch powodów:
- Te same typy obiektów (np. zmiennych) mogą mieć w różnych implementacjach różne wielkości.
- C++ pozwala użytkownikowi na definiowanie własnych typów obiektów. Często ważne jest, by wiedzieć ile pamięci zajmuje zdefiniowany obiekt.
- Operator sizeof ma następującą składnię:

sizeof (*nazwa_typu*)

albo

sizeof (*nazwa_obiektu*)

Wyrażenie warunkowe

w zależności od spełnienia lub niespełnienia warunku przyjmuje jedną z dwóch postaci:

(*warunek*) ? *wartość1* : *wartość2*

Przykładowo:

(i > 5) ? 15 : 20

Jeśli warunek jest spełniony, to wyrażenie przyjmuje wartość 15, natomiast jeśli warunek nie jest spełniony, to wyrażenie przyjmuje wartość 20.

Jest to bardzo wygodna konstrukcja, ponieważ pozwala zapakować ją do wnętrza innych instrukcji, np:

c = (x > y) ? 17 : 56;

Operator rzutowania

Operator rzutowania umożliwia przekształcenie typu obiektu. Działa on w ten sposób, że bierze obiekt jakiegoś typu i jako wynik zwraca obiekt innego typu. Operator ten może mieć jedną z dwóch postaci:

(*nazwa_typu*) *obiekt*

lub

nazwa_typu (*obiekt*)

Instrukcja warunkowa if

Instrukcja if może występować w jednej z dwóch postaci:

if (*wyrażenie*) *instrukcja1*;

Najpierw oblicza się wartość wyrażenia. Jeśli jest ono prawdziwe (różne od 0), to wykonywana jest *instrukcja1*. Jeśli wartość wyrażenia jest 0 (*fałsz*), to *instrukcja1* nie jest wykonywana.

if (*wyrażenie*) *instrukcja1*;
else *instrukcja2*;

Jeśli wartość wyrażenia jest 0, to wykonywana jest *instrukcja2*.

Wybór wielowariantowy

Możemy stosować wybór wielowariantowy używając zagnieżdżoną instrukcję if..else:

if (*warunek1*) *instrukcja1*;
else if (*warunek2*) *instrukcja2*;
else if (*warunek3*) *instrukcja3*;
.....;
else if (*warunekN*) *instrukcjaN*;

Zadanie 2

- Opracuj program obliczania podatku dochodowego według zasad podanych w tabeli. Użytkownik podaje podstawę obliczania podatku.

Podatek dochodowy od osób fizycznych

Skala podatkowa - 2007r

Podstawa obliczenia podatku (w zł)		Wartość podatku
poniżej	**43 405**	19% podstawy obliczenia minus kwota 572 zł 54 gr.
43 405	**85 528**	7 674 zł 41 gr. + 30% nadwyżki ponad 43.405 zł
85 528		20 311 zł 31 gr. + 40% nadwyżki ponad 85.528 zł

Kwota wolna od podatku **3.013 zł**

Instrukcja switch

Instrukcja switch służy do podejmowania wielowariantowych decyzji.

```
switch (wyrażenie)
{
   case wart1 :  instr1;
                 break;
   case wart2 :  instr2;
                 break;
   ...
   case wartn :  instrn;
                 break;
   default    :  instrn+1;
                 break;
}
```

```
/*-------------------------------------------------------------------------------------------*/
/* Program oblicza stopień na podstawie liczby otrzymanych punktów */
/* Kryteria: 0.. 49 pkt. - 2                                                                 */
/*           50.. 59 pkt. - 3                                                                */
/*           60.. 69 pkt. - 3.5                                                              */
/*           70.. 79 pkt. - 4                                                                */
/*           80.. 89 pkt. - 4.5                                                              */
/*           90..100 pkt. - 5                                                                */
/*-------------------------------------------------------------------------------------------*/
int main ()
{
  int lp;
  float stopien;

    cout << "Podaj liczbę punktów (0 <= lp <= 100): ";
    cin >> lp;
    lp = lp/10;
    switch (lp)
     {
     case 5    : { stopien = 3; break;}
     case 6    : { stopien = 3.5; break;}
     case 7    : { stopien = 4; break;}
     case 8    : { stopien = 4.5; break;}
     case 9,10 : { stopien = 5; break;}
     default   : { stopien = 2; break;}
     }
     cout << "Twoja ocena: ";
     cout.width(3);
     cout.precision(1);
     cout << stopien << endl;
     return 0;}
```

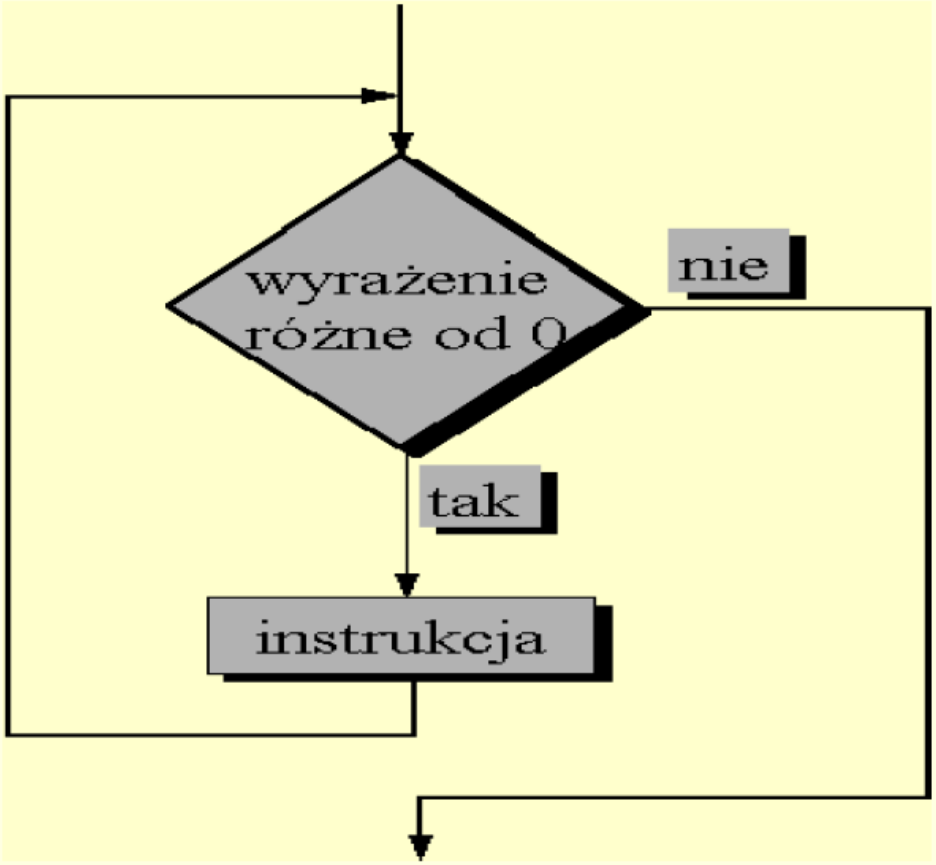
Instrukcje sterujące

Instrukcja while

while (*wyrażenie*) *instrukcja1*;

Najpierw obliczana jest wartość wyrażenia w nawiasach. Jeśli wartość ta jest *prawdziwa* (niezerowa), to następuje wykonywanie *instrukcji1* w pętli tak długo, aż wyrażenie przyjmie wartość *zerową* (fałsz). Należy zwrócić uwagę, że wartość wyrażenia jest obliczana *przed* wykonaniem instrukcji.

Instrukcja while

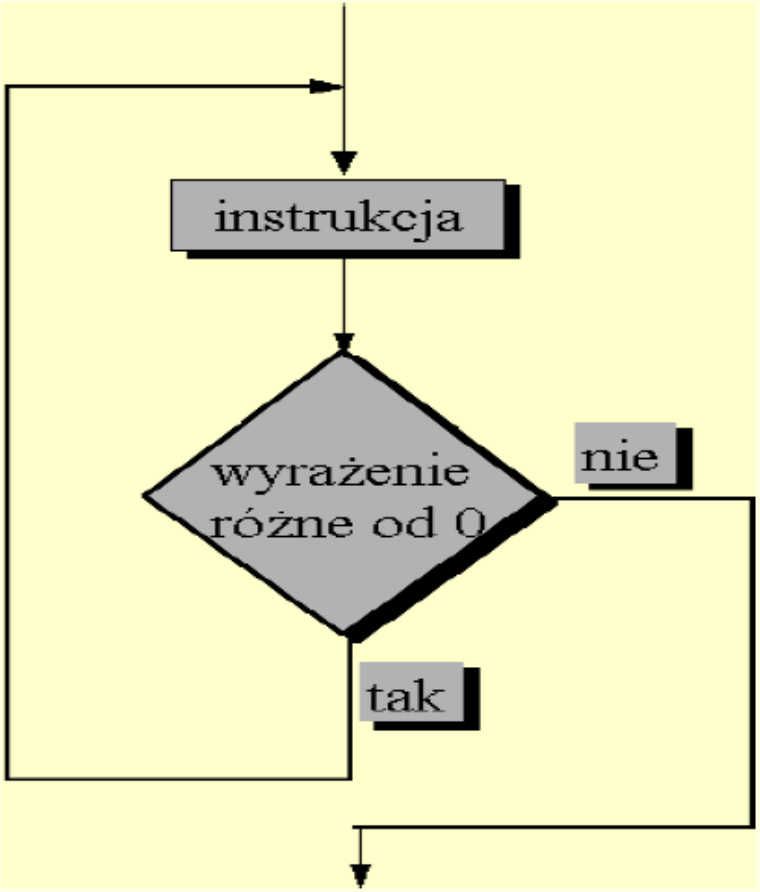


Instrukcja do...while...

do *instrukcja1* while (*wyrażenie*);

Działanie instrukcji: *instrukcja1* jest wykonywana
w pętli tak długo póki wyrażenie ma wartość *niezerową* (*prawda*). Z chwilą, gdy wyrażenie przyjmie wartość *zerową* (*fałsz*), działanie instrukcji zostaje zakończone.

Instrukcja do...while...



Instrukcja for

for (*instr_ini*; *wyraz_warunkowe*; *instr_krok*)
***treść_pętli*;**

instr_ini - jest to instrukcja wykonywana przed wykonaniem treści pętli;

wyraz_warunkowe - jest to wyrażenie obliczane przed każdym obiegiem pętli. Jeśli jest ono różne od zera, to wykonywane zostaną instrukcje będące treścią pętli;

instr_krok - jest to instrukcja wykonywana na zakończenie każdego obiegu pętli. Jest to ostatnia instrukcja wykonywana bezpośrednio przed obliczeniem wyrażenia warunkowego *wyraz_warunkowe;*

Działanie instrukcji for

1. najpierw wykonywana jest *instrukcja inicjalizująca* pracę pętli;
2. obliczane jest *wyrażenie warunkowe*; jeśli jest ono równe 0 - praca pętli jest przerywana;
3. jeśli *wyrażenie warunkowe* jest *różne od zera*, wówczas wykonywane zostaną instrukcje będące *treścią pętli*;
4. po wykonaniu treści pętli wykonana zostanie instrukcja *instr_krok*, po czym następuje powrót do p. 2.

Uwagi:

- *instr_ini* nie musi być tylko jedną instrukcją. *Może być ich kilka, wówczas muszą być one oddzielone przecinkami*. Podobnie jest w przypadku instrukcji *instr_krok*.
- Wyszczególnione elementy: *instr_ini*, *wyraz_warunkowe*, *instr_krok* nie muszą wystąpić. Dowolny z nich można opuścić, zachowując jednak średnik oddzielający go od sąsiada. Opuszczenie wyrażenia warunkowego jest traktowane tak, jakby stało tam wyrażenie *zawsze prawdziwe*.