Operatory w C++

- Operatory arytmetyczne
  - `+` dodawanie
  - `-` odejmowanie
  - `*` mnożenie
  - `/` dzielenie
  - `%` modulo
- Operatory relacyjne (porównania)
  - `<` mniejszy niż
  - `<=` mniejszy lub równy
  - `>` większy niż
  - `>=` większy lub równy
  - `==` równy
  - `!=` różny od
- Operatory logiczne
  - `!` negacja
  - `||` alternatywa
  - `&&` koniunkcja

Operatory w C++

- Złożone operatory przypisania

$+=$	$a+=b$	odpowiada	$a=a+b$
$-=$	$a-=b$	odpowiada	$a=a-b$
$*=$	$a*=b$	odpowiada	$a=a*b$
$/=$	$a/=b$	odpowiada	$a=a/b$
$\%=$	$a\%=b$	odpowiada	$a=a\%b$
$++$	$a++$	odpowiada	$a=a+1$
$--$	$a--$	odpowiada	$a=a-1$

- Operatory bitowe

$<<$	przesunięcie bitowe w lewo
$>>$	przesunięcie bitowe w prawo
$\vert$	bitowa alternatywa
&	bitowa koniunkcja
^	bitowa różnica symetryczna
$\sim$	bitowa negacja

Operator dzielenia

Wynik działania a/b zależy od typu argumentów a i b.

- Jeśli oba argumenty są liczbami całkowitymi, część ułamkowa jest odrzucana, a wynik jest liczbą całkowitą.
- Jeśli jeden z argumentów jest liczbą zmiennoprzecinkową, część ułamkowa jest zachowywana, a wynik jest zmiennoprzecinkowy.

```
#include <iostream>
using namespace std;
int main()
{
   cout << "9/5 = " << 9/5 << endl;
   cout << "9.0/5 = " << 9.0/5 << endl;
   cout << "9./5 = " << 9./5 << endl;
   cout << "9.0/5.0 = " << 9.0/5.0 << endl;
}
```

```
9/5 = 1
9.0/5 = 1.8
9./5 = 1.8
9.0/5.0 = 1.8
```

Operator modulo

- np. $15\%6 = 3$, bo $15 = 2 \cdot 6 + 3$.
- Argumenty operatora % muszą być liczbami całkowitymi.
- Jeśli jeden z argumentów jest ujemny, znak wyniku zależy od konkretnej implementacji.

```
#include <iostream>
using namespace std;
int main()
{
   cout << "6 % 4 = " << 6 % 4 << endl;
   cout << "-6 % 4 = " << -6 % 4 << endl;
   cout << "3 % 4 = " << 3 % 4 << endl;
   cout << "0 % 4= " << 0 % 4;
}
```

```
6 % 4 = 2
-6 % 4 = -2
3 % 4 = 3
0 % 4= 0
```

Operator modulo

Zamiana wagi w funtach na wagę w kamieniach (1 kamień = 14 funtów).

```
#include <iostream>
using namespace std;
int main()
{
   const int przelicznik = 14;
   int waga;
   cout << "Podaj wagę w funtach: ";
   cin >> waga;
   int kamienie = waga / przelicznik;
   int funty    = waga % przelicznik;
   cout << waga << " funt(y/ów) to " << kamienie << " kamieni,"
        << funty << " funt(y/ów).";
}
```

```
Podaj wagę w funtach: 177
177 funt(y/ów) to 12 kamieni, 9 funt(y/ów).
```

Operatory logiczne || && !

$5 == 5 \,\|\|\, 5 == 9$	prawda, bo prawdziwe jest pierwsze wyrażenie
$5 > 8 \,\|\|\, 5 < 10$	prawda, bo prawdziwe jest drugie wyrażenie
$5 < 8 \,\|\|\, 5 < 10$	prawda, bo oba wyrażenia są prawdziwe
$5 > 8 \,\|\|\, 5 > 10$	fałsz, bo oba wyrażenia są fałszywe

$5 == 5 \;\&\&\; 5 == 9$	fałsz, bo drugie wyrażenie jest fałszywe
$5 > 8 \;\&\&\; 5 < 10$	fałsz, bo pierwsze wyrażenie jest fałszywe
$5 < 8 \;\&\&\; 5 < 10$	prawda, bo oba wyrażenia są prawdziwe
$5 > 8 \;\&\&\; 5 > 10$	fałsz, bo oba wyrażenia są fałszywe

`if (x <= 5)`
można zapisać z użyciem operatora negacji:
`if (!(x > 5))`,
jednak pierwszy sposób jest czytelniejszy.

Operatory inkrementacji i dekrementacji

Operatory mają dwie formy: przedrostkową (++i) i przyrostkową (i++).

```
#include <iostream>
using namespace std;
int main()
{
   int i = 5;
   int j = 5;
   cout << "i   = " << i   << "     j = " <<   j << endl;
   cout << "i++ = " << i++ << "   ++j = " << ++j << endl;
   cout << "i   = " << i   << "     j = " <<   j;
}
```

```
i   = 5     j = 5
i++ = 5   ++j = 6
i   = 6     j = 6
```

Zapis i++ oznacza użycie w wyrażeniu aktualnej wartości *i*, a potem inkrementację tej zmiennej. Zapis ++i oznacza najpierw inkrementację zmiennej i, potem użycie w wyrażeniu nowej wartości.

Priorytety operatorów

Gdy w wyrażeniu występuje więcej niż jeden operator pojawia się pytanie, który z nich zostanie wykonany jako pierwszy.

```
int i = 3 + 4 * 5      // 35 czy 23?
```

- Liczba 4 wydaje się być argumentem obu operatorów: dodawania i mnożenia. W takim wypadku C++ stosuje *priorytety*.
- Operatory arytmetyczne mają priorytety podobne do zwykłych, algebraicznych zasad wykonywania działań, gdzie mnożenie, dzielenie i modulo są wykonywane przed dodawaniem i odejmowaniem.
- Mnożenie, dzielenie i modulo mają taki sam priorytet.
- Dodawanie i odejmowanie mają taki sam priorytet.

Łączność operatorów

Czasami same priorytety to za mało.

```
float f = 40 / 4 * 5;       // 50 czy 2?
```

- Ponownie 4 jest wspólnym argumentem dwóch operatorów. Jednak / i * mają taki sam priorytet.
- Wtedy C++ sprawdza, które operatory mają łączność od lewej do prawej, a które od prawej do lewej.
- Łączność **lewostronna** oznacza, że jeśli dwa operatory działające na tym samym argumencie mają ten sam priorytet, najpierw wykonywany jest operator znajdujący się po lewej stronie.
- Łączność **prawostronna** - odwrotnie.
- Mnożenie i dzielenie są łączne lewostronnie, więc wartość 4 zostanie najpierw użyta przez lewy operator (dzielenie).
- $40/4*5 = (40/4)*5$

Priorytety i łączność operatorów

Operatory relacyjne mają niższy priorytet niż operatory arytmetyczne.
$5 + 3 > 2 * 3 \equiv (5 + 3) > (2 * 3)$
Operatory logiczne mają niższy priorytet niż operatory relacyjne.
$5 > 3 \,||\, 4 < 6 \equiv (5 > 3) \,||\, (4 < 6)$
Operator ! ma priorytet wyższy od operatorów relacyjnych i arytmetycznych.
itd...

Choć priorytety operatorów w C++ umożliwiają pisanie złożonych wyrażeń bez nawiasów, to najlepiej jest używać nawiasów do grupowania warunków niezależnie od tego, czy nawiasy te są konieczne czy nie.
Ułatwia to czytanie kodu, nie wymaga uczenia się na pamięć priorytetów operatorów oraz ogranicza ryzyko popełniania błędów z powodu złego zapamiętania jakiegoś szczegółu.