Temat: Operatory bitowe, logiczne, przypisania, arytmetyczne, porównania (relacyjne), pozostałe (warunkowy, „„”, typeof, void, this, new, delete, In, instanceof). Priorytety operatorów.

Profil: Technik Informatyk (Technikum Informatyczne)
Klasa: 1
Przedmiot: E14.3

Spis treści

1 Operatory

Operatory w każdym języku programowania służą do operowania na wartościach i zmiennych. Przykładowo w wyrażeniu 4 + 5 znak + to właśnie operator.

1.1 bitowe

Pozwalają na wykonywanie operacji na poszczególnych bitach liczb. Są to:

- iloczyn bitowy (koniunkcja bitowa, operacja AND) &
- suma bitowa (alternatywa bitowa, operacja OR) |
- negacja bitowa (uzupełnienie do jedynki, operacja NOT) !
- oraz suma bitowa modulo 2 (alternatywa bitowa wykluczająca, różnica symetryczna, operacja XOR) oraz operacje przesunięć bitów.

Wszystkie operatory są dwuargumentowe, oprócz operatora bitowej negacji, który jest jednoargumentowy.

Operator	Użycie	Opis
`&`	`a & b`	AND. Zwraca jedynkę na każdej pozycji bitów, na której obiema argumentami są jedynki.
`|`	`a | b`	OR. Zwraca jedynkę na każdej pozycji bitów, na której przynajmniej jeden z argumentów jest jedynką.
`^`	`a ^ b`	XOR. Zwraca jedynkę na każdej pozycji bitów, na której jeden (i tylko jeden) z argumentów jest jedynką.
`~`	`~ a`	NOT. Odwraca bit podanego argumentu.
Przesunięcie w lewo	`a << b`	Przesuwa bity w `a` o `b` bitów w lewo wstawiając zera z prawej strony.
Przesunięcie w prawo	`a >> b`	Przesuwa bity w `a` o `b` bitów w prawo usuwając nadmiarowe bity z prawej strony.
Zero-fill right shift	`a >>> b`	Przesuwa bity w `a` o `b` bitów w prawo usuwając nadmiarowe bity z prawej strony i dodając zera z lewej.

A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A||B
0	0	0
0	1	1
1	0	1
1	1	1

A	![~]A
0	1
1	0

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

Przykłady:

```
15 & 9 = 9  (1111 & 1001 = 1001)
15 | 9 = 15 (1111 | 1001 = 1111)
15 ^ 9 = 6  (1111 ^ 1001 = 0110)
```

1.2 logiczne (boolowskie)

Można je wykonywać na argumentach, które posiadają wartość logiczną: prawda lub fałsz. W języku JavaScript wartości te są oznaczane jako true i false.

Operacje logiczne nie różnią się od tych znanych ze szkolnych lekcji matematyki. Są to AND (iloczyn logiczny), OR (suma logiczna) oraz NOT (negacja logiczna).

Operator	Opis	Przykład	Uwagi

`&&`	i	`x=3`
		`y=4`
		`((x < 9) && (y > 2))`
		`wynik:prawda`
			Iloczyn logiczny

`||`	lub	`x=3`
		`y=4`
		`((x==8) || (y==6))`
		`wynik:fałsz`
			Suma logiczna

!	zaprzeczenie	`x=3`
		`y=4`
		`!(x==y) wynik:prawda`
			Negacja logiczna

^	XOR	`x=2`
		`y=1`
		`((x==2) ^ (y>2))`
		`wynik:prawda`
			Sprawdza czy któryś z warunków jest spełniony przy czym nie mogą być spełnione oba naraz

1.3 przypisania

Operatory przypisania są dwuargumentowe i powodują przypisanie wartości argumentu znajdującego się z prawej strony operatora argumentowi znajdującemu się z lewej strony.

W języku JavaScript istnieje jednak również cały zestaw operatorów łączących operację przypisania z inną operacją — arytmetyczną, bitową lub łańcuchową. Przykładowo istnieje operator +=, który oznacza: przypisz argumentowi umieszczonemu z lewej strony wartość wynikającą z dodawania argumentu znajdującego się z lewej strony i argumentu znajdującego się z prawej strony operatora.

`liczba += 5` jest tłumaczony jako: `liczba = liczba + 5`

i oznacza: przypisz zmiennej liczba wartość wynikającą z dodawania liczba + 5.

Schematycznie można przedstawić ich znaczenie następująco: `arg1 op= arg2`
oznacza działanie `arg1 = arg1 op arg2`

Operator	Przykład	Równoważne z	Uwagi
=	x=y		
+=	x+=7	x = x + 7	
-=	x-=3	x = x - 3	
*=	x*=y	x = x * y	
/=	x/=y	x = x / y	
%=	x%=y	x = x % y	
<<=	y	x = x << y	
>>=	y	x = x >> y	
>>>=	y	x = x >>> y	>>> przesunięcie w prawo razem z bitem znaku
&=	y	x = x & y	& AND
|=	y	x = x | y	| OR
^=	y	x = x ^ y	^ XOR

1.4 arytmetyczne

Operatory *, /, +, -, % są dwuargumentowe, natomiast +, -, ++ i -- są jednoargumentowe.

Operator	Opis	Przykład	Wynik
+	Dodawanie	`x=3; x=x+4`	`7`
-	Odejmowanie	`x=4; x=6-x`	`2`
*	Mnożenie	`x=3; x=x*5`	`15`
/	Dzielenie	`10/5` `9/2`	`2` `4.5`

%	Modulo (reszta z dzielenia)	`4%3`
		`12%8`
		`8%2`
			`1`
			`4`
			`0`

++	Zwiększanie o 1	`x=2; x++`	`x=3`
--	Zmniejszanie o 1	`x=4; x--`	`x=3`
+	Ustalenie znaku wartości	`+x`	
-	Ustalenie znaku wartości	`-x`	

Modulo [%] reszta z dzielenia

Przykład:
10 % 3 = 1, trójka zmieści się bowiem w dziesięciu 3 razy, pozostawiając resztę 1 (3*3 = 9, 9 + 1 = 10).
21 % 8 = 5, gdyż 2 * 8 = 16, 16 + 5 = 21.

Operatory inkrementacji i dekrementacji

Oba mogą występować w formie przedrostkowej (np.: ++liczba) lub przyrostkowej (np.: liczba++).

Obie postacie powodują zwiększenie wartości zapisanej w *liczba* o jeden, ale w przypadku formy przedrostkowej (++liczba) odbywa się to przed wykorzystaniem zmiennej, a w przypadku formy przyrostkowej (liczba++) — dopiero po jej wykorzystaniu.

```
/*1*/ var liczba1 = 2;
/*2*/ document.write(liczba1++);
/*3*/ document.write(++liczba1);
/*4*/ document.write(liczba1);
/*5*/ var liczba2 = liczba1++;
/*6*/ document.write(liczba2);
/*7*/ liczba2 = ++liczba1;
/*8*/ document.write(liczba2);
/*9*/ document.write(++liczba2);
```

Po uruchomieniu kodu w oknie przeglądarki pojawi się ciąg liczb 244467. Dlaczego?

* 1 * Zadeklarowano zmienną liczba1 oraz przypisano jej wartość 2.
* 2 * Zastosowano formę przyrostkową operatora ++, zatem najpierw została wyświetlona wartość zmiennej liczba1 (liczba1 = **2**) na ekranie, a dopiero potem jej wartość zwiększyła się o jeden (liczba1 = 3).
* 3 * Odwrotnie, to znaczy, przez zastosowanie formy przedrostkowej najpierw zwiększa się wartość zmiennej liczba1 o jeden (liczba1 = **4**), a dopiero potem ta wartość zostaje wyświetlona na ekranie.
* 4 * Jedyną operacją jest ponowne wyświetlenie wartości liczba1 (liczba1 = **4**).
* 5 * Najpierw zostaje przypisana aktualna wartość liczba1 (liczba1 = 4) zmiennej liczba2 (liczba2 = 4), a dopiero potem zostaje zwiększona liczba1 o jeden (liczba1 = 5).
* 6 * Zostaje wyświetlona wartość liczba2 czyli **4**
* 7 * Najpierw jest zwiększana liczba1 o jeden (liczba1 = 6), a następnie tak powstała wartość jest przypisywana zmiennej liczba2 (liczba2 = 6).
* 8 * Zostaje wyświetlona wartość liczba2 na ekranie, czyli **6**.
* 9 * Najpierw zostaje zwiększona liczba2 o jeden (liczba2 = **7**), a dopiero potem ta wartość jest wyświetlona na ekranie.

Operator dekrementacji (--) działa analogicznie do ++, z tą różnicą, że zmniejsza wartość zmiennej o jeden.

1.5 porównywania (relacyjne)

Służą do porównywania argumentów. Wynikiem ich działania jest wartość logiczna `true` lub `false`, czyli prawda lub fałsz.

Operatory tego typu najczęściej wykorzystywane są w połączeniu z instrukcjami warunkowymi.

Operator	Opis	Przykład
==	jest równe	`2==3 wynik:fałsz`
===	identyczny, zwraca true jeżeli operandy są równe i tego samego typu.	`3 === var1`
!=	nie jest równe (różne)	`2!=3 wynik:prawda`
!==	nie identyczny, zwraca true jeżeli operandy nie są równe i/lub nie są tego samego typu.	`var1 !== 3` `3 !== '3'`
>	jest większe	`25>3 wynik:prawda`
<	jest mniejsze	`2<3 wynik:prawda`
>=	większe lub równe	`25>=3 wynik:prawda`
<=	mniejsze lub równe	`2<=3 wynik:prawda`

1.6 pozostałe

a) **operator warunkowy**

Pozwala on na szybkie sprawdzenie czy warunek jest prawdziwy, i zwrócenie jednej z dwóch podanych wartości.

Składnia: `warunek ? wartosc_1 : wartosc_2`
prawda fałsz

Jeżeli podany `warunek` będzie prawdziwy, wówczas zostanie zwrócona `wartosc_1`, natomiast w przeciwnym wypadku `wartosc_2`.

Przykład: Obliczenie wartości bezwzględnej liczby

```
var x=-5;
document.write(x >= 0 ? x : -x);
```

b) **przecinek**

Jego działanie polega na wyliczeniu wartości wyrażeń po obydwu jego stronach, i zwróceniu tej która jest po prawej stronie. Przydaje się to głównie w pętlach for, gdzie można go użyć do modyfikacji kilku wartości równocześnie.

```
for (n = 0, k = 3; n < 5; ++n, k += 4)
 document.write(n + k + "<br>");
```

Wynik:
```
3
8
13
18
23
```

c) **typeof()**

Operator jednoargumentowy. Sprawdza typ operandu.

Składnia: `typeof operand`
`typeof (operand)`

Zwraca: Łańcuch zawierający typ operandu: "number" "string", "boolean", "object", "function" i "undefined".
`Operand` jest łańcuchem znaków, zmienną, słowem kluczowym lub obiektem, którego typ ma zostać zwrócony. Zastosowanie nawiasów jest opcjonalne.

Przykład: `alert(typeof(1 + 2));`
Wynik: `number`

d) **void**

Operator jednoargumentowy. Najbardziej znanym zastosowaniem tego operatora jest dezaktywowanie akcji defaultowej dla HTMLowego elementu `<a>`.

Określa wyrażenie, które ma zostać ocenione bez zwracania wartości. Wyrażenie jest wyrażeniem JavaScript, które ma zostać ocenione. Nawiasy dokoła wyrażenia są opcjonalne, ale używanie ich jest w dobrym stylu.

Składnia: `void (wyrażenie)`
`void wyrażenie`

Możesz użyć operatora `void`, aby określić wyrażenie jako hipertekstowy odnośnik. Wyrażenie jest oceniane, ale nie jest ładowane w miejsce aktualnego dokumentu.

Poniższy kod tworzy hipertekstowy odnośnik, który nie wykonuje nic po kliknięciu go przez użytkownika. Gdy użytkownik kliknie odnośnik, `void(0)` zostanie oceniony jako 0, jednak nie ma to żadnego efektu w JavaScripcie.

```
<a href="javascript:void(0)">Kliknij tutaj, żeby nic się nie stało</a>
```

Dlaczego? Nie mam pojęcia jaka jest geneza użycia właśnie tego operatora tutaj. Być może zaleta szybkości, być może jakieś przesłanki historyczne. W praktyce ten sam efekt można uzyskać na wiele sposobów.

```
<a href=”javascript:undefined”>link</a>
<a href=”javascript:(function(){})()“>link</a>
```

lub nawet:

```
<a href=”javascript:void(123123123)“>link</a>
```

Wniosek – chcąc wyłączyć standardowe zachowanie linka wystarczy zwrócić tylko javascriptowo wartość `undefined`.

e) **this**

Stosuje się go najczęściej definiując obsługę zdarzeń:

Przykład:

```
<form name="formularz">
  <input type="text" name="text1" onChange="alert(this.value)"></input>
</form>
```

Po wpisaniu jakiegoś tekstu i naciśnięciu Enter na klawiaturze, pokarze się okienko alert z wpisanym tekstem.

Można go zastosować w obiektach, gdzie `this` odwołuje się do obiektu na którym pracujemy:

Przykład: Tworzenie własnego obiektu

```
function Samochod(producent, model, rok)
{
   this.producent = producent;
   this.model = model;
   this.rok = rok;
}
```

f) **new**

Tworzy nowe obiekty typu zdefiniowanego przez siebie, bądź predefiniowanego: `Array`, `Boolean`, `Date`, `Function`, `Image`, `Number`, `Object`, `Option`, `RegExp`, `String`. Na serwerze można użyć `DbPool`, `Lock`, `File`, `SendMail`.

Składnia: `objectName = new objectType ( param1 [,param2] ...[,paramN] )`

`objectName` Nazwa nowej instancji obiektu.
`objectType` Typ obiektu. Musi być funkcją definiującą typ obiektu.
`param1...paramN` Wartości własności obiektu. Własności te są parametrami zdefiniowanymi dla funkcji `objectType`.

Przykład:

```
var str = new String("Nowy napis");
document.write(str);
```

Przykład: Tworzenie własnego obiektu

```
function Samochod(producent, model, rok)
{
   this.producent = producent;
   this.model = model;
   this.rok = rok;
}

mojSamochod = new Samochod("Hennessey", "Venom GT", 2011);

document.write(mojSamochod.producent);
```

g) **delete**

Służy do usuwania elementu z tablicy, lub właściwości obiektu.

Usuwa zmienne zadeklarowane pośrednio, obiekty zdefiniowane przez użytkownika. Nie można usunąć zmiennych zadeklarowanych za pomocą słowa `var` ani też predefiniowanych.

Gdy użyje się `delete`, wtedy w miejsce usuwanego elementu jest wstawiana wartość `undefined`.

Składnia: `delete objectName`
`delete objectName[index]`

```
x=42
var y= 43
myobj=new Number()
myobj.h=4       // tworzymy własność h
delete x        // return true (można usunąć zmienną zadeklarowaną pośrednio)
delete y        // return false (nie można usunąć zmiennej zadeklarowanej za pomocą var)
delete Math.PI  // return false (nie można usunąć predefiniowanego obiektu)
delete myobj.h  // return true (można usunąć własność obiektu)
delete myobj    // return true (można usunąć obiekt zdefiniowany przez użytkownika)
```

Przykład:

```
var tab = new Array("ind1", "ind2", "ind3");
delete tab[1];
```

spowoduje usunięcie elementu tablicy o indeksie 1, numeracja pozostałych komórek pozostanie jednak bez zmian. Po wykonaniu tej operacji tablica `tab` będzie zawierała dwa elementy o indeksach 0 i 2. Operator `delete` dostępny jest w językach JavaScript od wersji 1.3.

Wywołanie: `document.write(tab[1]);`
Zwróci: `undefined`

h) **in**

Zwraca wartość `true` (prawda), jeśli określona własność jest w określonym obiekcie (czy w tablicy).

Składnia: `pole` **`in`** `objekt`

Przykład:

```
function Samochod(producent, model, rok)
{
   this.producent = producent;
   this.model = model;
   this.rok = rok;
}

mojSamochod = new Samochod("Hennessey", "Venom GT", 2011);

document.write("producent" in mojSamochod);     // true
```

i) **`instanceof`**

Sprawdza czy dany obiekt jest instancją określonego typu. Zwraca on wartość typu Boolean.

Składnia: `objectName` **`instanceof`** `objectType`

`objectName` Nazwa obiektu do porównania dla `objectType`.
`objectType` Typ obiektu.

Przykład:

```
kolor1 = new String("zielony");
document.write((kolor1 instanceof String) + "<br>"); // returns true

kolor2 = "niebieski";
document.write(kolor2 instanceof String); // returns false (kolor2
nie jest obiektem String)
```

Wynik:

```
true
false
```

1.7 Priorytety operatorów

Oprócz znajomości samych operatorów niezbędna jest jeszcze wiedza o tym, jaki mają one priorytety, czyli jaka jest kolejność wykonywania reprezentowanych przez nie operacji.

Wiemy np. z matematyki, że mnożenie jest silniejsze od dodawania, zatem najpierw mnożymy, potem dodajemy. W JavaScripcie jest podobnie, siła każdego operatora jest ściśle określona.

Im wyższa pozycja w tabeli, tym wyższy priorytet operatora. Operatory znajdujące się na jednym poziomie (w jednym wierszu) mają ten sam priorytet.

Lp.	Symbole	Operatory

1	`.`
	`()`
	`[]`
	`function()`
		Dostęp do pól
		Od najbardziej wewnętrznego do najbardziej zewnętrznego
		Indeks tablicy
		Wywołanie funkcji

2	`!`
	`~`
	`-`
	`++`
	`--`
	`delete`
	`new`
	`typeof`
	`void`
		Boolowskie Not
		Bitowe Not
		Zmiana znaku
		Inkrementacja, zwiększenie o 1
		Dekrementacja, zmniejszenie o 1
		Usunięcie składowej
		Utworzenie obiektu
		Ustalenie typu
		Wartość niezdefiniowana

3	`*  /  %`	mnożenie, dzielenie, reszta z dzielenia
4	`+  -`	dodawanie, odejmowanie
5	`<<  >>  >>>`	przesunięcie bitowe, w lewo, w prawo, w prawo z wypełnieniem zerami
6	`<  >  <=  >= instanceof`	mniejsze, większe, mniejsze lub równe, większe lub równe, sprawdzenie czy obiekt jest instancją klasy
7	`==  !=  ===  !==`	równe, różne, dokładna równość (identyczne), niedokładna równość
8	`&`	iloczyn bitowy
9	`^`	bitowa różnica symetryczna XOR
10	`|`	suma bitowa
11	`&&`	iloczyn logiczny
12	`||`	suma logiczna
13	`? :`	operator warunkowy

14	`=  +=  -=  *=  /=  %=`
	`&=  ^=  |=  <<=  >>=`
	`>>>=`
		operatory przypisania

15	`,`	rozdzielanie wyrażeń