Przeładowanie operatorów

Operator jako funkcja

- Operatory można traktować jako skrótowy zapis funkcji o odpowiedniej nazwie.
- W języku C++ obowiązuje umowa, że jeśli typ choć jeden z argumentów `x`,`y` w zapisie `x+y` jest klasą zdefiniowaną przez użytkownika, zapis ten jest traktowany jako skrócona forma wywołania funkcji
- `operator+(x,y)`.

  o nietypowej nazwie `operator+`.

- W analogicznej sytuacji zapis `x>=y` jest traktowany jako skrócona forma wywołania funkcji funkcji
- `operator>=(x,y)`.

Przeładowanie operatorów w języku C++

- Operatory w języku C++ mogą zostać przeładowane.
- Oznacza to, że mogą działać nie tylko na typach wbudowanych, ale również na obiektach klas zdefiniowanych przez użytkownika.
- Na przykład w klasie `szyld` możemy użyć operatora + do budowania nowych szyldów z połączenia (konkatenacji) już istniejących.

```
class szyld{
   ...
   friend szyld operator+(const szyld &pierwszySzyld,const
szyld &drugiSzyld);
   ...
}
...
szyld a("APTEKA"),b("DROGERIA");
szyld c,d;
...
c=a+b;
d=operator+(b,a); // To samo co d=b+a;
```

W języku C++ można przeładować wszystkie operatory z wyjątkiem

- :: (kwalifikator zakresu)

- . (wybór składowej)
- .* (wybór składowej za pomocą wskaźnika do składowej)
- ?: (wyrażenie warunkowe)

Wykaz operatorów w języku C++

Poniższa tabela zbiera informacje o operatorach stosowanych w C++:

Prio-ry-tet	Symbol	Ilość argu-mentów	Nazwa	Zastosowanie	Łącz-ność	Prze-łado-wanie
17	::	2	określenie zakresu	nazwa_klasy::składnik	L	nie
	::	1	nazwa globalna	::nazwa_globalna	P	nie
16	.	2	wybranie składnika nazwą	wskaźnik.składnik	L	nie
	->	2	wybranie składnika wskaźnikiem	wskaźnik->składnik		nie
	->	1	wygenerowanie wskaźnika z obiektu	obiekt->		
	[ ]	2	element tablicy	wskaźnik[wyrażenie]		
	( )	dowolna	wywołanie funkcji	funkcja(lista_argumentów)		
15	++	1	post-inkrementacja	Lwartość++	L	
	--	1	post-dekrementacja	Lwartość--		
	++	1	pre-inkrementacja	++Lwartość	P	
	--	1	pre-dekrementacja	--Lwartość		
	~	1	negacja bitowa	~wyrażenie		
	!	1	negacja	!wyrażenie		
	-	1	jednoargumentowy minus	-wyrażenie		
	+	1	jednoargument	+wyrażenie		

			owy plus			
	&	1	adres czegoś	&Lwartość		
	*	1	odniesienie się wskaźnikiem	*wyrażenie		
	new	1	rezerwuj pamięć	**new** typ		
	new[ ]	1	rezerwuj pamięć dla tablicy	**new** typ[*liczba_elementów*]		
	delete	1	zwolnij pamięć	**delete** wskaźnik		
	delete[ ]	1	zwolnij pamięć zarezerwowaną dla tablicy	**delete**[] wskaźnik		
( )	1	rzutowanie (konwersja typu)	(typ)wyrażenie			
14			Wybór składnika wskaźnikiem:			
	.*	2	i nazwą obiektu	obiekt.*wsk_do_składnika	L	nie
	->*	2	i wskaźnikiem do obiektu	wsk ->*wsk do_składnika		
13	*	2	mnożenie	wyrażenie*wyrażenie	L	
	/	2	dzielenie	wyrażenie/wyrażenie		
	%	2	modulo(reszta z dzielenia)	wyrażenie%wyrażenie		
12	+	2	dodawanie	wyrażenie+wyrażenie	L	
	-	2	odejmowanie	wyrażenie-wyrażenie		
11	<<	2	przesunięcie w lewo	Lwartość << wyrażenie	L	
	>>	2	przesunięcie w prawo	Lwartość >> wyrażenie		
10	<	2	mniejsze niż	wyrażenie < wyrażenie	L	
	<=	2	mniejsze lub równe	wyrażenie <= wyrażenie		
	>	2	większe od	wyrażenie > wyrażenie		

	>=	2	większe lub równe	wyrażenie >= wyrażenie		
9	==	2	równe	wyrażenie == wyrażenie	L	
	!=	2	nie równe	wyrażenie != wyrażenie		
8	&	2	iloczyn bitowy	wyrażenie & wyrażenie	L	
7	^	2	bitowa różnica symetryczna	wyrażenie ^ wyrażenie	L	
6	|	2	bitowa suma	wyrażenie | wyrażenie	L	
5	&&	2	koniunkcja	wyrażenie && wyrażenie	L	
4	||	2	alternatywa	wyrażenie || wyrażenie	L	
3	?:	3	arytmetyczne **if**	wyrażenie ? wyrażenie : wyrażenie	L	nie
2	=	2	zwykłe przypisanie	Lwartość = wyrażenie	P	
	*=	2	mnóż i przypisz	Lwartość *= wyrażenie		
	/=	2	dziel i przypisz	Lwartość /= wyrażenie		
	%=	2	modulo i przypisz	Lwartość %= wyrażenie		
	+=	2	dodaj i przypisz	Lwartość += wyrażenie		
	-=	2	odejmij i przypisz	Lwartość -= wyrażenie		
	<<=	2	przesuń w lewo i przypisz	Lwartość <<= wyrażenie		
	>>=	2	przesuń w prawo i przypisz	Lwartość >>= wyrażenie		
	&=	2	iloczyn bitowy i przypisanie	Lwartość &= wyrażenie		
	|=	2	suma bitowa i przypisanie	Lwartość |= wyrażenie		
	^=	2	bitowa różnica symetryczna i przypisanie	Lwartość ^= wyrażenie		

1	,	2	przecinek	wyrażenie , wyrażenie	L

Sposoby przeładowania operatorów

Operator można przeładować jako

- niestatyczną funkcję składową
- funkcję zaprzyjaźnioną z klasą
- zwykłą funkcję globalną

Operator przeładowany jako funkcja składowa ma na liście argumentów o jeden argument mniej niż powinien mieć. To dlatego, że tym brakującym (pierwszym) argumentem jest wskaźnik **this** do obiektu, na rzecz którego wywołano funkcję składową realizującą operator.

Na przykład

```
 class szyld
{
   ...
   szyld operator+(const szyld &drugiSzyld);
   ...
 }
 ...
 szyld a("APTEKA),b("DROGERIA");
 szyld c,d;
 ...
 c=a+b;
 d=b.operator+(a); // To samo co d=b+a;
```

Operator jako funkcja składowa czy zaprzyjaźniona z klasą?

Operator jako funkcja zaprzyjaźniona z klasą

- Najczęściej stosowane rozwiązanie.
- Tak pierwszy jak i drugi argument nie muszą być obiektami klasy. Wystarczy jeśli przy pomocy stosownego konstruktora można je skonwertować do obiektu danej klasy.

Operator jako niestatyczna funkcja składowa

- W rozwiązaniu tym pierwszy argument musi być obiektem danej klasy.
- Najczęściej przeładowujemy operator jako funkcję składową, jeśli modyfikuje ona swój pierwszy argument.

- Zwyczajowo ma to miejsce dla operatorów podstawienia (=,+=,-=,*=,/=) oraz operatorów dekrementacji i inkrementacji (--,++)
- Tylko ewentualny drugi argument nie musi być obiektem klasy, jeśli da się go skonwertować do obiektu klasy przez odpowiedni konstruktor.

Operator jako zwykła funkcja globalna

- Rozwiązanie stosowane tylko w przypadkach, gdy nie mamy dostępu bądź z jakiegoś powodu nie chcemy zmieniać kodu źródłowego klasy
- Musimy mieć wtedy dostęp do składników prywatnych klasy poprzez funkcje składowe.

Operatory: =,[],(),-> i ->* mogą być przeładowane tylko jako niestatyczne funkcje składowe.

Przeładowanie operatorów << i >>

Operatory << i >> w swojej wersji podstawowej wykonują cykliczne przesunięcie lewego argumentu o ilość bitów podaną w prawym argumencie odpowiednio w prawo lub w lewo.

W klasach **ostream** i **istream** standardowej biblioteki C++ operatory te zostały przeładowane tak by umożliwić pisanie do strumienia i czytanie ze strumienia dowolnych typów wbudowanych.

Pisząc zaprzyjaźnioną z klasą **samochod** funkcję

```
    friend ostream& operator<<(ostream& out, samochod
jakisSamochod)
{
        out << "Samochod " << jakisSamochod.model <<
               " o przebiegu " << jakisSamochod.przebieg;
        return out;
};
```

uzyskujemy prosty sposób wypisywania do strumienia obiektów klasy **samochod**.

Kiedy stosować przeładowanie operatorów?

- Główna korzyść jaka płynie ze stosowania operatorów, to uproszczenie zapisu skomplikowanych formuł.
- Jeśli zdefiniujemy klasę **macierz**, to bez przeładowania operatorów wyrażenie (A+B)*(C+D) musimy zapisać mniej więcej tak:
- ```
  iloczyn(suma(A,B),suma(C,D));
  ```
  ```

co jest o wiele mniej czytelne.

- Przeładowanie operatorów jest wspaniałym mechanizmem dla klas opisujących takie obiekty, w których stosowanie operatorów jest naturalne.
- Dlatego najczęściej stosuje się je do obiektów matematycznych takich jak wektor, macierz, liczba zespolona.
- Stosowanie przeładowania operatorów tam gdzie nie ma naturalnych skojarzeń z danym operatorem nie jest dobrą praktyką, bo szybko zapomina się jaką funkcję dany operator ma wypełniać.

- Na przykład w klasie `szyld` czy `duzySzyld` `operator` + kojarzy się w naturalny sposób z łączeniem dwóch szyldów w jeden.
- Jednak trudno znaleźć naturalne skojarzenie na przykład dla operatora -.