Podstawy programowania

Podstawowe elementy języka C+ +

- Program komputerowy w języku C++ składa się m.in. ze zbioru plików tekstowych zawierających kod źródłowy z deklaracjami (ang. *declarations'),* definicjami (ang. *definitions),* wyrażeniami (ang. *expressions),* instrukcjami (ang. *statements)* i innymi konstrukcjami tego języka, np. komentarzami (ang. *comments).* Wspomniane pliki tekstowe to pliki źródłowe (ang. *source files)* i pliki nagłówkowe (ang. *header files).* Zostały omówione, podobnie jak deklaracje, definicje, wyrażenia, instrukcje itd., w dalszej części tego podrozdziału.
- W kodzie źródłowym programu wykorzystuje się różne słowa. Znaczenie niektórych z nich jest zastrzeżone dla określonych elementów składowych języka. Są to słowa kluczowe *(words).* Na przykład słowo kluczowe if to nazwa instrukcji, słowo kluczowe double to nazwa typu danych, a true to wartość logicznej prawdy.

Podstawowe elementy języka C+ +

- Pozostałych słów można używać do nadawania nazw — tzw. **identyfikatorów** (ang. *identi- fiers)* — elementom składowym programu zdefiniowanym przez programistę: stałym, zmiennym, funkcjom, klasom, obiektom itp. Identyfikator w C++ składa się z dowolnego ciągu liter, cyfr oraz kresek podkreślenia (ang. *underscores)* i musi rozpoczynać się od litery. Na przykład poprawnymi identyfikatorami w języku C++ są: boki, pole Prostokąta, dane wejściowe.
- Kod źródłowy programu komputerowego w języku C++ musi być zgodny z określonymi regułami (zasadami) syntaktycznymi ustalonymi dla tego języka. Przy czym wspomniane reguły syntaktyczne dotyczą składni języka, czyli jego „gramatyki" i „ortografii". Na przykład ważną regułą składniową jest zasada, że każdą instrukcję w języku C++ kończy się średnikiem (;).

Podstawowe elementy języka C+ +

- W kodzie źródłowym oprócz słów kluczowych, reprezentujących zastrzeżone elementy języka, oraz identyfikatorów skojarzonych z elementami programu zdefiniowanymi przez programistę mogą się znaleźć tzw. **komentarze** (ang. *comments)*. Komentarze to informacje dla programisty (programistów) dotyczące różnych aspektów kodu — począwszy od opisu ogólnej koncepcji zastosowanego rozwiązania, a skończywszy na szczegółowych informacjach związanych z pojedynczymi instrukcjami.
- W języku C++ można stosować komentarze dwojakiego rodzaju:
- **komentarze jednoliniowe** (ang. *single-line comments),*
- **komentarze wieloliniowe (ang.** *multi-line comments).*
- Komentarz jednoliniowy obejmuje ciąg znaków od symbolu //do końca linii, np.

  //tojest komentarz jednoliniowy
- Komentarz wieloliniowy zaś rozpoczyna się symbolem /*, a kończy */ i może obejmować wiele linii, np.

  /' To jest komentarz wieloliniowy, który może obejmować wiele linii.

  '/

Zmienne

- **Zmienne** (ang. *variables)* są wykorzystywane w programie do przechowywania w pamięci operacyjnej komputera wartości, które mogą się zmieniać w trakcie jego działania. Każda zmienna w programie powinna być zadeklarowana.
- Postać ogólna **deklaracji zmiennej** (ang. *variable declaration)* jest następująca: typ_zmiennej nazwa_znriennej; gdzie:
- typ_zmiennej — typ danych (ang. *data type),* do którego należy zmienna, np. double,
- nazwa.zmlennej — identyfikator zmiennej, np. boki.

Jeśli trzeba zadeklarować kilka zmiennych należących do tego samego typu, należy użyć składni:

typ_zmiennej nazwa_zmiennej_l, nazwa_znriennej_2, nazwa_znnennej_3;

gdzie nazwa_zmiennej_l, nazwa_zmi ennej_2, nazwa_zmi ennej_3 to identyfikatory zmiennych.

Typy danych

- Każda zmienna zadeklarowana w programie może przechowywać wartości typu określonego w jej deklaracji. Typy danych w języku C++ można podzielić umownie na:
- typy predefiniowane,
- typy zdefiniowane przez użytkownika.
- **Predefiniowane typy danych** (ang. *predefined data types)* to typy wewnętrzne kompilatora (ang. *compiler interna! data types)*. Dlatego też nazywa się je często **typami wbudowanymi** (ang. *built-in data types)*. Do wbudowanych typów danych języka C++ należą:
- typy całkowite,
- typy zmiennoprzecinkowe,
- typ znakowy,
- typ logiczny,
- typ void.

Typy danych

- Z wbudowanych typów danych można korzystać w programie bez potrzeby ich definiowania. Do **typów danych zdefiniowanych przez użytkownika** (ang. *user-defined data types)* zalicza się wszystkie pozostałe typy danych, które albo określa samodzielnie programista, albo zostały zdefiniowane wcześniej przez innego programistę (programistów) i są zawarte np. w konkretnej bibliotece. Aby można było w programie skorzystać ze zdefiniowanego typu danych, należy zapewnić dostęp do jego definicji w sposób bezpośredni lub za pośrednictwem np. wspomnianej wcześniej biblioteki.
- Inny podział typów danych wynika z rodzaju „bazy" będącej podstawą konstrukcji określonego typu danych. Według tego kryterium typy danych w języku C++ można podzielić na:
- typy podstawowe (pierwotne),
- typy pochodne.

Typy danych

- **Podstawowe typy danych** *(ang.fundamental data types)* są typami elementarnymi. Zawierają one określone zbiory wartości, których nie można podzielić na elementy składowe. Do podstawowych typów danych należą wszystkie typy wbudowane, które wymieniono wcześniej.
- Typy podstawowe są stosowane jako bazy konstrukcyjne lub elementy składowe innych — **pochodnych typów danych** (ang. *derived data types)*. Dlatego typy podstawowe nazywa się **również typami pierwotnymi** (ang. *primitive data types)*.

Typy danych

- **Typy całkowite**
- Typy całkowite (ang. *integer types)* umożliwiają przechowywanie w zmiennych wartości liczb całkowitych. Typ danych o nazwie int pełni funkcję podstawowego (bazowego) typu całkowitego.
- Na przykład deklaracja zmiennej o nazwie zmiennalnt typu całkowitego int ma postać: int zmi ennalnt;. W procesie kompilacji programu kompilator zarezerwuje w pamięci operacyjnej dla tej zmiennej co najmniej 4 bajty. Zakres (ang. *scope)* dopuszczalnych wartości, jakie może przyjąć zmienna typu i nt, obejmuje przedział liczbowy od -2 147 483 648 do +2 147 483 647.
- Typ bazowy int może być modyfikowany. Służą do tego modyfikatory (ang. *modifiers)*: short, long, signed i unsigned. W wyniku użycia nazwy typu bazowego int wraz z odpowiednim modyfikatorem lub połączeniem wybranych modyfikatorów można uzyskać inne typy danych całkowitych. Typy te różnią się od siebie zakresem dopuszczalnych wartości oraz ilością miejsca zajmowanego w pamięci operacyjnej. Na przykład użycie modyfikatora long określa typ long int, a połączenie modyfikatorów unsigned i long daje w rezultacie typ unsigned long int.

Typy danych

- **Typy zmiennoprzecinkowe**

Typy zmiennoprzecinkowe (ang. *floating-point types)* są również nazywane **typami zmiennopozycyjnymi.** Pozwalają one na przechowywanie w zmiennych wartości należących do zbioru liczb rzeczywistych. Do typów zmiennoprzecinkowych zalicza się; float, double oraz long double. W zmiennych typu float można zapamiętać liczby o pojedynczej precyzji (ang. *Single precisiori),* double — podwójnej precyzji (ang. *doubleprecision),* long double — rozszerzonej precyzji (ang. *Extended precision)*. Stąd zmienna typu float zajmuje w pamięci operacyjnej 4 bajty, zmienna typu double — 8 bajtów, a typu long double — 10 bajtów. Nq przykład deklaracja zmiennej o nazwie (identyfikatorze) zmienna Double typu double ma postać: double zmienna Double;.

- **Typ znakowy**

Typ znakowy (ang. *character type)* jest oznaczany za pomocą słowa kluczowego char. Jest on wykorzystywany w deklaracjach zmiennych, w których pamiętane są liczby całkowite z przedziału od -128 do 127 lub pojedyncze znaki z przedziału od 0 do 255. Wraz z nazwą typu char można używać modyfikatorów signed oraz unsigned, np. unsigned char. Zmienne typu char są przechowywane w pamięci operacyjnej w obszarach (blokach) o pojemności 1 bajta.

Typy danych

- **Typ logiczny**

Nazwa wbudowanego typu logicznego (ang. *boolean type)* to bool. Do typu bool należą dwie predefiniowane wartości: true (prawda) oraz false (fałsz). Zmienna typu bool zajmuje w pamięci 1 bajt.

- **Typ void**

Typ void reprezentuje zbiór pusty, czyli taki, który nie zawiera żadnych wartości.

- **Przykład**

char znak;

long int boki, bok2;

double a, b, c;

W przedstawionym kodzie zadeklarowano zmienną o nazwie znak należącą do predefiniowanego typu podstawowego char. W następnej linii zadeklarowano dwie zmienne typu long i nt. Identyfikator pierwszej z nich to boki, a drugiej — bok2. W ostatniej linii zadeklarowano zmienne a, b i c należące do typu double.

Typy danych

- **Aliasy nazw predefiniowanych typów danych**
- W języku C++ programista ma możliwość określenia własnych nazw (identyfikatorów) dla predefiniowanych typów danych podstawowych. Innymi słowy, można tworzyć aliasy (ang. *aliases)* nazw, czyli inne, zastępcze nazwy dla już istniejących typów danych. Można to zrealizować zasadniczo na dwa sposoby:

przy użyciu **specyfikatora** (ang. *specifier)* typedef,

za pomocą **słowa kluczowego** (ang. *keyword)* usi ng.

Ogólna postać użycia specyfikatora typedef jest następująca:

Typedef typ_istniejący alias_nazwy;

gdzie typ_istniejącyjest nazwą istniejącego, predefiniowanego typu danych, a alias_nazwy to jego inna, nowa, zastępcza nazwa.

Ogólna składnia użycia słowa kluczowego usi ng, która pozwala określić alias nazwy istnie-jącego typu danych, ma postać:

using alias_nazwy = typ_istniejący;

gdzie alias.nazwy i typ_istniejący mają takie samo znaczenie jak w przypadku słowa kluczowego typedef.

Typy danych

- **Aliasy nazw predefiniowanych typów danych**

- **Przykład**
- typedef unsigned char byte; using word = unsigned int;
- byte zmienna Byte; word zmienna Word;
- W kodzie zdefiniowano aliasy nazw typów danych: unsi gned char — jako byte, oraz unsi gned int —jako word. W pierwszym przypadku wykorzystano słowo kluczowe typedef, a w drugim using. Następnie zadeklarowano dwie zmienne całkowite: zmienna Byte i zmienna Word. Pierwsza z nich należy do typu byte, a druga do typu word.

Typy danych

- **Aliasy nazw predefiniowanych typów danych**

- **Przykład**
- typedef unsigned char byte; using word = unsigned int;
- byte zmienna Byte; word zmienna Word;
- W kodzie zdefiniowano aliasy nazw typów danych: unsi gned char — jako byte, oraz unsi gned int —jako word. W pierwszym przypadku wykorzystano słowo kluczowe typedef, a w drugim using. Następnie zadeklarowano dwie zmienne całkowite: zmienna Byte i zmienna Word. Pierwsza z nich należy do typu byte, a druga do typu word.

Inicjalizacja zmiennych

- Zmiennej w programie można nadać wartość początkową już podczas jej deklarowania. Operacja ta nazywa się **inicjalizacją zmiennej** (ang. *variable initialization)*. Ogólna postać deklaracji zmiennej należącej do jednego z typów podstawowych połączona z jej inicjalizacją jest następująca:
- typ_zmiennej nazwa_zmiennej = wyrażenie;

lub

- typ.zmiennej nazwa_zmiennej (wyrażenie);

lub

- typ_zmiennej nazwa_zmiennej {};
- typ.zmiennej nazwa_zmiennej {wyrażenie};
- typ_zmiennej nazwa_zmiennej = {wyrażenie};

gdzie:

- typ.zmiennej — typ danych, do którego należy zmienna,
- nazwa_zmiennej —identyfikator zmiennej,
- wyrażenie (ang. *expression)* — sekwencja zmiennych, stałych i operatorów zgodna z regułami języka C++, określająca operacje, które dają w rezultacie wartość typu zgodnego Z typem.zmiennej.

Inicjalizacja zmiennych

Pierwszy z wymienionych wcześniej sposobów inicjalizacji zmiennej został odziedziczony po języku C. Dlatego też można go nazwać **C-inicjalizacją** (ang. *C-like initialization)*. Sposób ten jest również nazywany **inicjalizacją kopiującą** (ang. *copy initialization)*.

Drugi sposób jest nazywany **inicjalizacją bezpośrednią** (ang. *direct initialization)* albo **inicjalizacją konstruktorową** (ang. *constructor initialization)*.

Trzeci sposób, wprowadzony w specyfikacji C++11, jest nazywany **inicjalizacją jednolitą** (ang. *uniform initialization)* lub **inicjalizacją klamrową** (ang. *brace initialization)*. Inicjalizacja jednolita może występować w programie w różnych postaciach (formach):

Jeżeli podczas inicjalizacji zmiennej metodą jednolitą zostanie użyta postać: typ_zmiennej nazwa,zmiennej {};, zmienna zostanie zainicjowana wartością zerową (lub pustą) w zależności od typu_zmiennej. Taka inicjalizacja jest nazywana **inicjalizacją zerową** (ang. *zero initialization)*.

- Kolejna postać inicjalizacji jednolitej, typ-zmiennej nazwa_zmiennej {wyrażenie};, odpowiada **inicjalizacji jednolitej bezpośredniej (ang.** *direct uniform initialization)*.
- Ostatnia postać, typ_zmiennej nazwa_zmiennej = {wyrażenie};, to **inicjalizacja jednolita kopiująca** (ang. *copy uniform initialization)*.

Inicjalizacja zmiennych

- **Przykład**

```
int zmienna1 = 1;
int  zmienna2  (zmiennal +     i);
int  zmienna3  {};
int  zmienna4  {zmienna3 +     i};
int  zmienna5  = {zmienna4     + i};
```

- We fragmencie kodu zadeklarowano pięć zmiennych. Każda z nich została zainicjowana podczas deklaracji. Zmiennej zmienna1 nadano wartość początkową równą 1 przy użyciu inicjalizacji kopiującej. Zmienna zmienna2 została zainicjowana wartością wyrażenia zmienna1 + 1, które daje w rezultacie wartość 2. W tym przypadku zastosowano inicjalizację bezpośrednią konstruktorową. Zmiennej zmienna3 nadano wartość początkową równą 0, co wynika z wykorzystanej inicjalizacji zerowej. Zmienną zmienna4 zainicjowano wartością wyrażenia zmienna3 + *1 (- 2)* przy użyciu inicjalizacji jednolitej bezpośredniej. Ostatnią zmienną (zmienna5) zainicjowano wartością wyrażenia zmienna4 + 1, które daje w rezultacie liczbę 2 — za pomocą inicjalizacji jednolitej kopiującej. W przykładzie wykorzystano operator dodawania arytmetycznego +.

Deklaracja zmiennej z dedukcją typu

- Deklaracja zmiennej może być połączona z dedukcją typu danych (ang. *data type deduction),* do którego kompilator zaliczy tę zmienną. W tym celu wykorzystuje się słowa kluczowe decltype oraz auto.
- Specyfikator decltype może być użyty w deklaracji zmiennej w celu wydedukowania jej typu na podstawie typu innej zmiennej lub typu wyniku zadanego wyrażenia. Postać ogólna użycia specyfikatora decltype jest następująca:
- decltype(zmienna_wzorcowa) nazwa_ztniennej ;

lub

- decltype(wyrażenie) nazwa.zmiennej;

gdzie:

- nazwa_zmiennej — identyfikator deklarowanej zmiennej,
- zmi enna_wzorcowa — zmienna, na podstawie której zostanie wydedukowany typ zmiennej deklarowanej,
- wyrażenie — wyrażenie, na podstawie którego kompilator wydedukuje typ deklarowanej zmiennej.

Deklaracja zmiennej z dedukcją typu

- **Przykład**

```
int zmiennal {1};
decltype(zmiennal),zmienna2;' '
zmienna2 =2;
decltype(zmienna2 + 1) zmienna3;
zmienna3 = 3;
```

- W przedstawionym fragmencie programu zadeklarowano zmienną o nazwie zmi ennal typu całkowitego int. Zmienna ta została zainicjowana wartością i. Następnie została zadeklarowana zmienna zmienna2 — ale bez inicjalizacji. Jej typ (int) został wydedukowany przez kompilator na podstawie typu zmiennej zmienna1. Zmiennej tej nadano w osobnej instrukcji wartość równą 2. Na końcu zadeklarowano zmienną ó nazwie zmienna3. Typ zmiennej zmienna3 (int) został wydedukowany na podstawie wyniku wyrażenia zmienna2 + 1.

Stałe

- W ujęciu ogólnym **stałe** (ang. *constants)* to wyrażenia (ang. *expressions)* mające ustaloną wartość, która się nie zmienia w trakcie działania programu. Na przykład stałą jest liczba całkowita o wartości dziesiętnej równej i, jak również wyrażenie 1 + 1.

Literały

- Wyrażenia określające wartości stałych w kodzie źródłowym programu można formułować przy użyciu **literałów** (ang. *literals)*. W zależności od typu wyrażanej wartości literały można podzielić na:
- całkowite,
- zmiennoprzecinkowe,,
- znakowe,
- łańcuchowe,
- logiczne.

Literały

- **Literały całkowite** (ang. *integer literals)* definiują stałe całkowite. Literały całkowite można zapisywać w systemie dziesiętnym (ang. *decimal),* ósemkowym (ang. *octal)* i szesnastkowym (ang. *hexadecimal),* zakończone opcjonalnie przyrostkiem — modyfikatorem (ang. *modifier)* U (lub u), L (lub 1) albo LL (lub 11). Wybrane modyfikatory można ze sobą łączyć.
- Na przykład *25* to literał całkowity zapisany w systemie dziesiętnym, 0!5 — literał całkowity zapisany w systemie ósemkowym, 0x!5 — literał całkowity zapisany w systemie szesnastkowym, 35L — literał całkowity zapisany w systemie dziesiętnym traktowany jako należący do typu long (zamiast domyślnie i nt), 25UL — literał całkowity zapisany w systemie dziesiętnym traktowany jako unsigned long.
- **Literały zmiennoprzecinkowe** (ang. *floating point literals)* wyrażają liczby rzeczywiste. Literały zmiennoprzecinkowe można zapisywać w dwojaki sposób: przez podanie albo jedynie części całkowitej i ułamkowej liczby, np. 1.2345, albo dodatkowo wykładnika potęgi liczby 10, oznaczanego jako e lub E. Na przykład literał i.2345e-s oznacza liczbę $1{,}2345 \cdot 10^{-5}$.

Literały zmiennoprzecinkowe mogą być zakończone modyfikatorami przyrostkowymi F (lub f) oraz u (lub u). Na przykład literał 1.2345F jest traktowany przez kompilator jako należący do typu float zamiast (domyślnie) do typu double.

Literały

- **Literały znakowe** (ang. *character literals)* reprezentują pojedyncze znaki. Literały znakowe zapisuje się w pojedynczych apostrofach (ang. *single quotes),* ', np. ' A', ' a ', ';'.

Do literałów znakowych zalicza się również **znaki specjalne** (ang. *special characters)*. Są one poprzedzone znakiem \ (ang. *backslash)*. Znaki specjalne mają przypisane określone znaczenie. Na przykład znak ■ \n' oznacza nową linię, a ' \t' to znak tabulacji.

- **Literały łańcuchowe** (ang. *string literals)* obejmują napisy ujęte w podwójne apostrofy (ang. *double quot es),* '', np. "język C++", "C".
- **Literały logiczne** (ang. *logical literals)* reprezentują określoną wartość logiczną: albo prawdę — true, albo fałsz — false.

Stałe nazwane

- W programowaniu często się zdarza, że trzeba wiele razy użyć pewnej ustalonej wartości, która w przypadku ogólnym może być określona za pomocą wyrażenia. Wówczas można za każdym razem wpisywać do kodu to wyrażenie albo przypisać mu unikatowy identyfikator i zamiast wyrażenia wykorzystywać ten zdefiniowany identyfikator.
- Stała, która została skojarzona z wybranym unikatowym identyfikatorem, to **stała nazwana** (ang. *named constant)*. Ogólna postać definicji takiej stałej jest następująca:
- const typ_stałej nazwa_stałej   = wyrażenie;

lub

- const typ_stałej nazwa.stałej   (wyrażenie);

lub

- const typ_stałej nazwa_statej {wyrażenie};

gdzie:

- nazwa_stalej — identyfikator stałej,
- typ_stale j — typ danych, do którego należy stała,
- wyrażenie — dowolne wyrażenie, którego wartość kompilator skojarzy z nazwa_stałej.

Operatory

- **Operator** (ang. *operator)* stanowi symbol jedno- lub wieloznakowy, który informuje kompilator, jaką operację ten ma wykonać, np. operację dodawania arytmetycznego dwóch liczb. Argumentami — **operandami** (ang. *operands)* — operatorów mogą być zmienne oraz stałe. Język C++ oferuje wiele **operatorów wbudowanych** (ang. *built-in operators),* które można podzielić na kilka grup:
- operatory przypisania,
- operatory arytmetyczne,
- operatory bitowe,
- operatory porównania,
- operatory logiczne
- i inne.

Operatory przypisania

- Zadaniem **operatorów przypisania** (ang. *assignment operators)* jest modyfikacja wartości zmiennej. Język C++jest wyposażony w: • .
- operator przypisania prostego,
- operatory przypisania złożonego. .

Operator przypisania prostego (ang. *simple assignment operator)* umożliwia „przypisanie" wartości określonego wyrażenia do zmiennej. Postać ogólna wyrażenia zawierającego operator przypisania prostego jest następująca:

- nazwa_zmiennej = wyrażenie;

lub

- nazwa_zmiennej = (wyrażenie);

Lub

- nazwa_zmiennej = {wyrażenie};

gdzie nazwa_zmiennej oznacza zmienną, której należy przypisać wartość wyniku wyrażeni a z prawej strony operatora. Typ wyniku wyrażenia powinien być zgodny z typem zmiennej.

Operatory przypisania

Operatory przypisania złożonego (ang. *compound assignment operators)* z kolei pozwalają zmodyfikować wartość zmiennej przez wykonanie na niej określonej operacji, np. operacji dodania do niej wartości innej zmiennej. Jest to równoważne przypisaniu wyniku wykonywanej operacji do modyfikowanej zmiennej. Wybrane operatory przypisania złożonego przedstawiono w tabeli

Tabela 3.1. Wybrane operatory przypisania złożonego

Nazwa	Symbol	Przykład	Przykład równoważny
Przypisanie dodawania	+ x	zmienna += 10	zmienna = zmienna + 10
Przypisanie odejmowania		zmienna -= 10	zmienna = zmienna - 10
Przypisanie mnożenia	* =	zmienna *= 10	zmienna = zmienna * 10
Przypisanie dzielenia	/=	zmienna /= 1	zmienna = zmienna / 10
Przypisanie modulo	%=	zmienna 5K= 1	zmienna = zmienna % 10

Operatory arytmetyczne

- **Operatory arytmetyczne** (ang. *arithmetic operators)* mają za zadanie obliczenie wyniku określonej operacji arytmetycznej. Można je podzielić w zależności od liczby argumentów na:
- • jednoargumentowe, • dwuargumentowe.
-

Operatory arytmetyczne

- Wykaz **operatorów jednoargumentowych** (ang. *unary operators)* przedstawiono w tabeli
- Operatory arytmetyczne jednoargumentowe

Nazwa	Symbol	Przykład
Operator znaku plus	+	+a
Operator znaku minus	-	-a
Operator inkrementacji		++
preinkrementacja		++zmienna
postinkrementacja		zmienna++
Operator dekrementacji		
predekrementacja		—zmienna
postdekrementacja		zmienna—

Jeżeli operator **preinkrementacji** (ang. *pre-incrementation)* lub **predekrementacji** (ang. *pre- -decrementation)* danej zmiennej jest częścią składową wyrażenia, to wartość tego wyrażenia jest obliczana dopiero po zwiększeniu lub zmniejszeniu o 1 wartości zmiennej. W przypadku **postinkrementacji** (ang. *post-incrementation)* i postdekrementacji (ang. *post-decrementation)* sytuacja jest odwrotna — najpierw obliczana jest wartość wyrażenia, a dopiero po zakończeniu tej operacji wartość zmiennej jest zwiększana lub zmniejszana o 1.

Operatory arytmetyczne

- **Operatory dwuargumentowe** „operują" na dwóch argumentach (operandach), np. zmienna 1 + zmienna2. Wykaz operatorów dwuargumentowych zawiera tabela

Nazwa	Symbol	Przykład
Operator dodawania		a+b
Operator odejmowania		a-b
Operator mnożenia	*	a*b
Operator dzielenia	/	a/b
Operator modulo	%	%b

- W *ogólności typy argumentów* operatorów wymienionych w tabeli *mogą być* zarówno rzeczywiste (np. float, double), jak i całkowite (np. int, long). Jednakże operator *modulo,* który pozwala na obliczenie reszty z dzielenia, wymaga użycia argumentów (operandów) całkowitych.
- W danym wyrażeniu może występować wiele operatorów (w tym operatorów arytmetycznych) oraz wiele argumentów (operandów), na których są wykonywane zadane operacje. Na przykład w wyrażeniu w = ++a + b S c; występują trzy operatory: inkrementacji++, dodawania + i modulo %, oraz trzy argumenty: a, b i c.

Priorytety operatorów

- Operatory, które odpowiadają za wykonanie określonych operacji, mają różny **priorytet** (ang. *operator priority)* — czyli **pierwszeństwo w wykonaniu** (ang. *operatorprecedence)*. Operatory o wyższym priorytecie mają pierwszeństwo w wykonaniu nad operatorami o niższym priorytecie. Na przykład operatory inkrementacji i dekrementacji mają wyższy priorytet od operatorów mnożenia, dzielenia i modulo, a operatory mnożenia, dzielenia i modulo mają wyższy priorytet od operatorów dodawania i odejmowania. Tym samym operacje inkrementacji i dekrementacji zostaną wykonane przed operacjami mnożenia, dzielenia i modulo, a operacje mnożenia, dzielenia i modulo zostaną wykonane przed operacjami dodawania **i** odejmowania. '
- Jeżeli w danym wyrażeniu występują wyłącznie operatory o tym samym priorytecie (np. operatory mnożenia i dzielenia), to działania są wykonywane w kolejności od lewej do prawej strony tego wyrażenia.

Priorytety operatorów

- **Przykład 3.10**
- int a = 99, b = 20, c = 3;
- int w = ++a + b % c;
- Po zadeklarowaniu i inicjalizacji zmiennych a, b i c powinna zostać obliczona wartość wyrażenia ++a + b % c. W tym celu na początku zostanie poddana inkrementacji zmienna a. Po tej operacji wartość a wynosi 100. Następnie zostanie obliczona wartość wyrażenia b % c. Wynik tej operacji to 2 (reszta z dzielenia całkowitego 20 / 3). Kolejna operacja to dodawanie 100 + 2, która daje w rezultacie wartość 102. Wartość ta zostaje na końcu przypisana do zmiennej w.

Operatory bitowe

- **Operatory bitowe** (ang. *bitwise.operatorś)* pozwalają na wykonywanie operacji na poszczególnych bitach liczb całkowitych. Podobnie jak operatory arytmetyczne, można je podzielić na jedno- i dwuargumentowe.

Nazwa	Symbol	Przykład	Opis
Operator bitowy AND	&	liczba1 & liczba2	Wykonanie iloczynu logicznego AND na odpowiadających sobie bitach dwóch liczb
Operator bitowy OR	|	liczba1 | liczba2	Wykonanie sumy logicznej OR na odpowiadających sobie bitach dwóch liczb
Operator bitowy XOR	^	liczba1 ^ liczba2	Wykonanie sumy logicznej XOR na odpowiadających sobie bitach dwóch liczb
Operator przesunięcia w lewo o *k* pozycji	<<	Liczba << k	Wykonanie przesunięcia każdego bitu liczby zadaną liczbę *k* pozycji w lewo
Operator przesunięcia w prawo o *k* pozycji	>>	Liczba >> k	Wykonanie przesunięcia każdego bitu liczby zadaną liczbę *k* pozycji w prawo
Operator bitowy NOT	-	- liczba	Wykonanie negacji logicznej NOT na każdym bicie liczby

Operatory bitowe - przykład

Int liczba = 7; *//0111* int wynik;

wynik = liczba & liczba; *//0111*

wynik = liczba | liczba; *//0111*

wynik = liczba ^ liczba; *//0000*

wynik = -liczba; *//-1000*

wynik = liczba « 1; *//1110* wynik = liczba >> 1; *//0011*

Na początku zadeklarowano i zainicjowano wartością 7 zmienną liczba. W kodzie binarnym liczba 7 to om. Następnie zadeklarowano zmienną wynik, w której zapisywane będą kolejno rezultaty wykonania poszczególnych operacji bitowych. Bitowe AND przyjmuje na danej pozycji wartość i wtedy i tylko wtedy, gdy odpowiadające sobie bity mają wartości równe i. Bitowe AND dla dwóch identycznych argumentów liczba daje w wyniku liczbę 7 w systemie dziesiętnym (binarnie om). Wartość bitowego OR to również 7 w systemie dziesiętnym. Wynika to z faktu, że bitowe OR przyjmuje na danej pozycji wartość 0 wtedy i tylko wtedy, gdy odpowiadające sobie bity mają wartości równe o. W przeciwnym razie bitowe OR na tej pozycji osiąga wartość i. Wartość bitowego XOR dla dwóch argumentów liczba wynosi o, co wynika z faktu, że wynik XOR na danej pozycji jest równy 1 wtedy i tylko wtedy, gdy odpowiadające sobie bity w argumentach są różne. Wynik bitowego NOT to dziesiętne -8 (w systemie binarnym -1000). Operator jednoargumentowy bitowego NOT odwraca wszystkie bity w liczbie będącej jej argumentem. Wynik przesunięcia w lewo o jedną pozycję dla argumentu liczba daje w rezultacie liczbę 14 w systemie dziesiętnym (binarnie 1110). Przesunięcie w prawo o jedną pozycję daje wynik 3 dziesiętnie (0011 binarnie).

Operatory porównania

Operatory porównania (ang. *comparison operators)* są używane **w** celu porównania wartości dwóch wyrażeń. Do operatorów porównania zalicza się:

operatory równości (ang. *equality operators),*

operatory relacyjne (ang. *relational operators).*

Nazwa	Symbol	Przykład
Operator „równy"	==	Zmienna 1 == zmienna 2
Operator „różny"	!=	Zmienna 1 != zmienna 2
Operator „mniejszy"	<	Zmienna 1 < zmienna 2
Operator „większy"	>	Zmienna 1 > zmienna 2
Operator „mniejszy lub równy"	>=	Zmienna 1 <= zmienna 2
Operator „większy lub równy"	-	Zmienna 1 >= zmienna 2

Wynik wykonania operacji porównania wartości dwóch argumentów (operandów) należy do typu logicznego i wynosi albo true, albo false.

Operatory porównania

```
int zmienna1 = 1;
int zmienna2 = 2;
bool wynik;
wynik = zmienna1 == zmienna2;
wynik = zmienna1 < zmienna2;
```

Po zadeklarowaniu i zainicjowaniu zmiennych całkowitych zmienna1 i zmienna2 następuje deklaracja zmiennej logicznej o nazwie wynik. Kolejna linia kodu zawiera wyrażenie zmienna1 == zmienna2, w którym za pomocą operatora równości == następuje sprawdzenie, czy zmienne zmienna1 i zmienna2 są równe. Wynik tego wyrażenia (false) jest zapisywany w zmiennej wynik. W ostatniej linii kodu został wykorzystany operator < (mniejszy). Wynik wyrażenia zmienna1 < zmienna2 (true) jest podstawiany do zmiennej wynik.

Operatory logiczne

Operatory logiczne (ang. *logical operators)* operują na argumentach (operandach) typu logicznego bool, które mogą być stałymi i zmiennymi. W ogólności argumentami mogą być wyrażenia dające w wyniku albo wartość logiczną true, albo false

Nazwa	Symbol	Opis
Operator NOT (negacja)	!	Zwraca wartość logiczną false, jeśli argument jest prawdą (true), i odwrotnie
Operator AND (iloczyn logiczny)	&& and	Zwraca wartość true wtedy i tylko wtedy, gdy oba argumenty są prawdą (true). W przeciwnym razie zwraca wartość false
Operator OR (suma logiczna)	|| or	Zwraca wartość true wtedy, gdy co najmniej jeden argument jest prawdą (true). W przeciwnym razie zwraca wartość false

Operatory logiczne - przykład

```
int zmienna1 = 1;
int zmienna2 = -1;
bool wynik;
wynik = !(zmienna1 < zmienna2);
wynik = (zmienna1 > 0) and (zmienna2 > 0);
wynik = (zmienna1 > 0) || (zmienna2 > 0);
```

Na początku zadeklarowano i zainicjowano zmienne zmienna1 i zmienna2. Następną instrukcję stanowi deklaracja zmiennej logicznej wynik. W czwartej linii wykorzystano operator negacji !. Po tym wyznaczana jest wartość relacji zmienna1 < zmienna2. Jej wynik to false. Negacja false to true. I ta wartość jest podstawiana do zmiennej wynik. W kolejnej linii kodu najpierw wyznaczane są wartości wyrażeń: zmienna1 > 0 (true) i zmienn2 > 0 (false). Taka kolejność wynika z zastosowania nawiasów (), które zmieniają kolejność wykonywania operacji. Następnie obliczany jest iloczyn logiczny z dw6ch wartości wyrażeń obliczonych wcześniej. Jego wynik to false. W ostatniej linii następuje obliczenie sumy logicznej ||.

Jej wynik to true, ponieważ wartość jednego z wyrażeń wynosi true.

Operator sizeof

Operator sizeof zwraca rozmiar (w bajtach), jaki zajmuje w pamięci operacyjnej określony typ danych, stała, zmienna lub w ogólności wyrażenie. Ogólna postać użycia operatora sizeof jest następująca:

sizeof (typ_danych)

lub

sizeof wyrażenie

gdzie typ_danych oznacza typ danych (podstawowy lub zdefiniowany przez programistę), a wyrażenie jest dowolnym wyrażeniem w języku C++ zawierającym stałe, zmienne i operatory.

Operator sizeof - przykład

```
const int stała = 1;
int zmienna = 1;
int rozmiar;

// Określenie rozmiaru typu int:
rozmiar = sizeof(int); //4
// Określenie rozmiaru stałej stała:
rozmiar = sizeof stała; //4
// Określenie rozmiaru zmiennej zmienna:
rozmiar = sizeof zmienna; //4
// Określenie rozmiaru wyrażenia (stała + zmienna):
rozmiar = sizeof (stała + zmienna); //4
```

Wyrażenia

wyrażenie to sekwencja zmiennych, stałych i operatorów zgodna z regułami języka C++, określająca obliczenia (operacje), które dają w rezultacie wartość określonego typu. Wyrażenie składa się z operandów (argumentów) połączonych za pomocą operatorów. Na przykład wyrażeniem jest suma dwóch liczb całkowitych a + b lub warunek (a > o) && (b > o).

Wyrażenia można podzielić w zależności od typu ich wyniku (wyznaczonej wartości) na:

- **wyrażenia stałe** (ang. *constant expres$ions),* **np.** 1 + 10,
- **wyrażenia całkowite** (ang. *integer expressions),* np. 10 * a, gdzie a należy do typu int,
- **wyrażenia rzeczywiste** (ang. *float expressionś),* np. 10 / a, gdzie a należy do typu float,
- **wyrażenia bitowe** (ang. *bttwise expressions),* np. a | b, gdzie a i b należą do typu i nt,
- **wyrażenia porównania** (ang. *comparison expressions),* **np.** a > o, gdzie a należy do typu int,
- **wyrażenia logiczne** (ang. *logical expre$sions),* np. a && b, gdzie a i b należą do typu bool.

Konwersja typu

Konwersja typu (ang. *type corwersion)* oznacza zmianę typu. W czasie konwersji tworzona jest nowa wartość pewnego typu z wartości innego typu. W ogólności konwersji typu mogą podlegać stałe, zmienne i wyrażenia. Konwersja typu może się odbywać w sposób niejawny („po cichu”) lub jawny. Z tego wynikają rodzaje konwersji:

konwersja niejawna,

konwersja jawna.

Konwersja niejawna

- **Konwersja niejawna** (ang. *implicit conversion)* jest realizowana przez kompilator w sposób automatyczny — „po cichu", bez użycia jakichkolwiek jawnych (widocznych w kodzie) poleceń jej wykonania. Dlatego ten rodzaj konwersji jest również nazywany **konwersją automatyczną** (ang. *automatic conversion)*. Podczas konwersji automatycznej konwertowana wartość jest kopiowana w sposób niejawny do innego typu, który jest kompatybilny z typem początkowym. Na przykład w instrukcji long zmiennaLong = 10; najpierw w sposób automatyczny realizowana jest niejawna konwersja wartości 10 z typu -i nt na (kompatybilny z typem -int) typ long, a dopiero później-ta skonwertowana wartość jest przypisywana do zmiennej zmiennaLong. Konwersja niejawna może być realizowana przez kompilator na typach danych podstawowych, np. int, long, float, double. Stąd ten rodzaj konwersji ma jeszcze jedną nazwę: konwersja standardowa (ang. *standard conversion)*.

Konwersja niejawna

Konwersja standardowa zwykle zachodzi wtedy, gdy w danym wyrażeniu występuje więcej niż jeden typ danych. Jeżeli konwersja jest realizowana z typu „mniejszego" na typ „większy", np. z typu int na typ long, to operacja ta nazywa się promocją typu (ang. *typepromotion)*. W promocji typu nie są tracone żadne informacje, np. precyzja. Jeżeli natomiast kompilator jest zmuszony wykonać konwersję z typu „większego" na typ „mniejszy", np. z typu double na typ float lub typ i nt, niektóre informacje, np. część ułamkowa liczby, mogą zostać utracone.

Przykład

```
// Deklaracja zmiennej zmiennaLong typu long int:
long zmiennaLong;
// Definicja stałej stalaInt:
const int stalaInt = 1;
//Nadanie wartości zmiennej zmiennaLong:
zmiennaLong = stalaInt; //1
```