3.2.1. Struktura programu

Program w języku C++ zbudowany jest z dwóch części:

- część deklaracyjna,
- część operacyjna.

Część deklaracyjna obejmuje wszystkie deklarowane elementy programu, które mogą zostać wykorzystane w części operacyjnej. Ta część służy do definiowania bądź deklarowania różnych elementów, jednak nie można ich w tym miejscu uruchomić. Do tego wykorzystuje się część operacyjną.

Część operacyjna to program główny, który w tym języku jest funkcją o nazwie `main()`. Po uruchomieniu programu wykonywane są wszystkie polecenia zawarte w funkcji `main()`. W tej części programu korzystamy ze zdefiniowanych lub zadeklarowanych przez nas elementów znajdujących się w części deklaracyjnej.

W początkowym etapie nauki języka C++ będziemy korzystać z uproszczonej budowy programu. W następnych podrozdziałach poznasz bardziej zaawansowane możliwości konstruowania programu.

Przykład 3.1.

Przyjrzyjmy się konstrukcji pierwszego programu (*prog3_1.cpp*), który został przedstawiony w tabeli 3.2. Komentarze umieszczone z prawej strony wyjaśniają znaczenie poszczególnych elementów implementacji. W języku C++ **rozróżniane są małe i wielkie litery**, kod programu należy więc zapisywać dokładnie tak, jak podano w podręczniku.

Tabela 3.2. Pierwszy program w języku C++

Kod programu	Komentarz
CZĘŚĆ DEKLARACYJNA	
`#include <iostream>`	Dyrektywa preprocesora — **preprocesor** to program dokonujący wstępnego przeglądu kodu źródłowego programu **Deklaracja bibliotek** (czyli plików nagłówkowych), z których korzystamy w programie Biblioteka `iostream` zawiera definicje standardowego strumienia wejścia `cin` i standardowego strumienia wyjścia `cout`
`using namespace std;`	Deklaracja korzystania z **przestrzeni nazw standardowych** `std` (czyli biblioteki standardowej)

CZĘŚĆ OPERACYJNA	
`main()`	Nagłówek funkcji głównej
`{`	
`int a, b, c;`	Deklaracja zmiennych lokalnych a, b, c
`a = 2;`	Przypisanie zmiennej a wartości 2
`cout<<"podaj wartość zmiennej b:";`	Komunikat wypisany na ekranie „podaj wartość zmiennej b:"
`cin>>b;`	Wczytanie z klawiatury wartości zmiennej b
`c = a + b;`	Przypisanie zmiennej c sumy zmiennych a i b
`cout<<a<<" + "<<b<<" = "<<c<<endl;`	Wypisanie wyniku na ekranie
`return 0;`	Przypisanie funkcji `main()` wartości 0
`}`	

Wskazówka

Nagłówek funkcji głównej `main()` w pełnej postaci zapisywany jest następująco: `int main (void)`, gdzie:

- `int` oznacza, że typ wartości zwracanej z funkcji `main()` jest całkowity, funkcji tej przypisywana jest więc wartość typu `int`, pominięcie typu w nazwie funkcji oznacza, że jest ona typu `int`;
- `main` to nazwa funkcji;
- `void` oznacza „pusty", a w tym przypadku brak parametrów, które wpisuje się w nawiasie, pusty nawias jest równoważny `void`.

Programy przedstawione w tym podręczniku napisane zostały w **trybie konsolowym** (czyli tekstowym). Z tego wynika, że po uruchomieniu programu w środowisku Windows, otwarte zostanie okienko konsoli, w którym wykonywany jest program. Aby okienko konsoli nie zamknęło się zaraz po zakończeniu programu, musimy zastosować odpowiednie działania. Poniżej podano polecenia, które w tej sytuacji można wykorzystać.

Jeśli pracujemy w systemie Windows, wystarczy w funkcji `main()` przed zakończeniem programu wpisać polecenie `system("pause")`, co spowoduje uruchomienie funkcji systemowej `pause`. Program będzie wyglądał wówczas następująco (*prog3_2.cpp*):

```
#include <iostream>
using namespace std;
main()
{
 int a, b, c;
 a = 2;
 cout<<"podaj wartość zmiennej b:";
 cin>>b;
```

```
 c = a + b;
 cout<<a<<" + "<<b<<" = "<<c<<endl;
 system("pause");
 return 0;
}
```

Można skorzystać również z polecenia **cin.get()** (patrz tabela 3.5). Jednak aby ta funkcja działała poprawnie, należy wcześniej wyczyścić bufor klawiatury poleceniem **fflush(stdin)** lub zignorować znaki znajdujące się w strumieniu wejściowym poleceniem **cin.ignore()** (patrz tabela 3.5). Przedstawiony program ma więc jedną z następujących postaci (*prog3_3.cpp*, *prog3_4.cpp*):

```
#include <iostream>
using namespace std;
main()
{
 int a, b, c;
 a = 2;
 cout<<"podaj wartość zmiennej b:";
 cin>>b;
 c = a + b;
 cout<<a<<" + "<<b<<" = "<<c<<endl;
 cin.ignore();
 cin.get();
 return 0;
}
```

```
#include <iostream>
using namespace std;
main()
{
 int a, b, c;
 a = 2;
 cout<<"podaj wartość zmiennej b:";
 cin>>b;
 c = a + b;
 cout<<a<<" + "<<b<<" = "<<c<<endl;
 fflush(stdin);
 cin.get();
 return 0;
}
```

W przypadku pominięcia deklaracji przestrzeni nazw standardowych **std** operacje wejścia i wyjścia dostępne są dopiero po podaniu nazwy klasy **std**, czyli **std::cout**, **std::cin**. Program z takimi zmianami przedstawiono poniżej (*prog3_5.cpp*):

```
#include <iostream>
main()
{
 int a, b, c;
 a = 2;
 std::cout<<"podaj wartość zmiennej b:";
 std::cin>>b;
 c = a + b;
 std::cout<<a<<" + "<<b<<" = "<<c<<std::endl;
 return 0;
}
```

3.2.2. Operacje wejścia-wyjścia

Strumieniowe operacje wejścia-wyjścia dostępne są po dołączeniu biblioteki `iostream`. Korzystać możemy z dwóch standardowych strumieni:

- `cout` — standardowy strumień wyjściowy, który zwykle skojarzony jest z monitorem, stosujemy tutaj operator kierunkowy wstawiania znaków do strumienia wyjściowego: `<<`;
- `cin` — standardowy strumień wejściowy, który zwykle skojarzony jest z klawiaturą, wykorzystujemy w tym przypadku operator kierunkowy pobierania danych ze strumienia do podanej zmiennej: `>>`.

Wypisując komunikat poleceniem `cout`, w łańcuchu oznaczanym cudzysłowem można stosować **znaki specjalne**. W tabeli 3.3 podano zestawienie podstawowych znaków specjalnych.

Tabela 3.3. Zestawienie znaków specjalnych

Znak specjalny	Działanie
`\n`	*new line* (przejście do nowej linii)
`\t`	*tab* (tabulator poziomy)
`\v`	*vertical tab* (tabulator pionowy)
`\b`	*backspace* (przesunięcie kursora o jeden znak w lewo)
`\r`	*carriage return* (przesunięcie kursora na początek bieżącego wiersza)
`\'`	' (wypisanie znaku zastrzeżonego — apostrof)
`\"`	" (wypisanie znaku zastrzeżonego — cudzysłów)
`\?`	? (wypisanie znaku zastrzeżonego — znak zapytania)
`\\`	\ (wypisanie znaku zastrzeżonego — *backslash*)

Przykład 3.2.

Przyjrzyjmy się przykładom zastosowania strumieniowych operacji wejścia i wyjścia przedstawionym w tabeli 3.4. Z prawej strony umieszczono komentarze wyjaśniające wykonywane operacje.

Tabela 3.4. Przykłady zastosowania operacji wejścia i wyjścia

Przykładowe operacje wejścia i wyjścia	Efekt działania polecenia	Komentarz
`cout<<"tekst"<<" "<<"tekst";`	*tekst tekst*	Nie ma potrzeby wydzielania poszczególnych słów wypisywanego tekstu, taki sam efekt uzyskamy poleceniem `cout<<"tekst tekst";`
`cout<<"tekst\ttekst\ntekst";`	*tekst tekst* *tekst*	Wewnątrz wyświetlanego łańcucha można stosować znaki specjalne
`int a=5, b=3;` `cout<<"a="<<a<<"\n"<<a+b;`	*a=5* *8*	W strumieniu wyjściowym można umieszczać zarówno komunikaty, jak i zmienne, które należy od siebie oddzielać operatorem <<, znak specjalny `\n` stosowany pojedynczo można zapisywać `"\n"` lub `'\n'`
`int a;` `cin>>a;`	Oczekiwanie na wpisanie z klawiatury wartości zmiennej a i potwierdzenie tego klawiszem *Enter*	Wczytanie z klawiatury wartości zmiennej a
`int a, b, c;` `cin>>a>>b>>c;`	Oczekiwanie na wpisanie z klawiatury wartości zmiennych a, b i c oddzielonych spacjami lub *Enterem* oraz potwierdzenie tego klawiszem *Enter*	Przy wczytywaniu z klawiatury wartości wielu zmiennych wygodnie jest wykonać to za pomocą jednej operacji `cin`, wczytywane zmienne mogą być różnego typu

Dodatkowe funkcje realizujące strumieniowe operacje wejścia przedstawione zostały w tabeli 3.5, natomiast funkcje dotyczące operacji wyjścia — w tabeli 3.6.

Tabela 3.5. Funkcje realizujące operacje wejścia ze strumienia

Funkcja	Opis funkcji	Przykładowe zastosowanie funkcji	Znaczenie
`int get(void)`	Pobiera następny znak ze strumienia	`char a;` `a = cin.get();`	Wczytanie z klawiatury wartości zmiennej a, równoważne: `cin>>a;`
`get(char &z)`	Pobiera ze strumienia znak i przypisuje go zmiennej z	`char a;` `cin.get(a);`	Wczytanie z klawiatury wartości zmiennej a, równoważne: `cin>>a;`
		`cin.get();`	Wymuszenie naciśnięcia klawisza *Enter*
`get(char *tekst, int dl, char k='\n')`	Odczytuje znaki ze strumienia do zmiennej `tekst` do momentu osiągnięcia znaku `k` (znak ten nie jest odczytywany), odczytania `dl`–1 znaków lub odczytania wszystkich znaków	`char s[50];` `cin.get(s,sizeof(s),'a');`	Wczytywanie z klawiatury do zmiennej `s` znaków, aż do pojawienia się znaku „a", odczytania 49 znaków lub odczytania wszystkich znaków
		`cin.get(s,24);`	Wczytywanie z klawiatury do zmiennej `s` znaków, aż do pojawienia się znaku specjalnego `'\n'` (domyślnie), odczytania 23 znaków lub odczytania wszystkich znaków
`getline(char *tekst, int dl, char k='\n')`	Odczytuje znaki ze strumienia (również spacje) do zmiennej `tekst` do momentu osiągnięcia znaku `k` (znak ten jest pobierany ze strumienia, ale nie jest dopisywany do zmiennej `tekst`), odczytania `dl`–1 znaków lub odczytania wszystkich znaków	`char s[50];` `cin.getline(s,40,'c');`	Wczytywanie z klawiatury do zmiennej `s` znaków, aż do pojawienia się znaku „c" lub odczytania 39 znaków
		`cin.getline(s,sizeof(s))`	Wczytywanie z klawiatury do zmiennej `s` wszystkich znaków (maksymalna długość łańcucha wynosi 256 znaków)
		`cin.getline(s,50)`	Wczytywanie z klawiatury do zmiennej `s` znaków, aż do odczytania 49 znaków
		`string s1;` `getline(cin,s1);`	Wczytywanie z klawiatury do zmiennej `s1` wszystkich znaków (maksymalna długość łańcucha wynosi 256 znaków)
`ignore()`	Powoduje pominięcie znaków znajdujących się w strumieniu wejściowym	`cin.ignore();`	Znaki znajdujące się w strumieniu wejściowym zostają zignorowane

Tabela 3.6. Funkcja realizująca operacje wyjścia ze strumienia

Funkcja	Opis funkcji	Przykładowe zastosowanie funkcji	Znaczenie
`put(char &z)`	Wstawia do strumienia znak `z`	`char a='p'; cout.put(a);`	Wypisanie na ekranie wartości zmiennej `a`, równoważne: `cout<<a;`

Do formatowania strumieniowych operacji wejścia i wyjścia najczęściej wykorzystuje się **manipulatory**. Dostępne są one po dołączeniu biblioteki `iomanip`. Najważniejsze manipulatory przedstawione zostały w tabeli 3.7.

Tabela 3.7. Zestawienie manipulatorów strumieniowych

Manipulator	Działanie
`dec`	Włącza konwersję dziesiętną
`hex`	Włącza konwersję szesnastkową
`oct`	Włącza konwersję ósemkową
`endl`	Umieszcza w strumieniu znak końca wiersza i opróżnia strumień
`ends`	Umieszcza znak `'\0'` na końcu łańcucha
`setfill(char c)`	Ustala znak wypełniania wolnych miejsc pola na `c`
`setprecision(int n)`	Określa precyzję liczb rzeczywistych równą `n`
`setw(int n)`	Wyznacza szerokość pola równą `n` znaków

Przykład 3.3.

Przyjrzyj się przykładom zastosowania manipulatorów strumieniowych w przedstawionym poniżej programie (*prog3_6.cpp*):

```
#include <iostream>
#include <iomanip>
using namespace std;
main()
{
 int k=241;
 cout<<"k = "<<k<<endl;
 cout<<"konwersja systemów liczbowych:"<<endl;
 cout<<"szesnastkowy = "<<hex<<k<<endl;
 cout<<"dziesiętny = "<<dec<<k<<endl;
 cout<<"szesnastkowy = "<<setfill('0')<<setw(6)<<hex<<k<<endl;
 cout<<"ósemkowy = "<<setfill('*')<<setw(10)<<oct<<k<<endl;
 return 0;
}
```

Powyższy program wypisuje wartość zmiennej k w różnych systemach liczbowych z podaną szerokością pola i znakiem wypełniania. Efekt działania tego programu jest następujący:

k = 241

konwersja systemów liczbowych:

szesnastkowy = f1

dziesiętny = 241

szesnastkowy = 0000f1

*ósemkowy = *******361*

Przykład 3.4.

Przykład pokazuje zastosowanie manipulatorów dla zmiennych typu rzeczywistego (*prog3_7.cpp*):

```
#include <iostream>
#include <iomanip>
using namespace std;
main()
{
 double x=22.111111111, y=3.5555555555555, w, z;
 cout<<"podaj liczbę rzeczywistą: ";
 cin>>w;
 cout<<"\nx = "<<setw(10)<<x<<endl;
 cout<<"\ny = "<<setfill('0')<<setw(12)<<y<<endl;
 cout<<"\ny = "<<setprecision(8)<<setfill('0')<<setw(12)<<y<<endl;
 cout<<"\nw = "<<setfill('*')<<setw(15)<<w<<endl;
 cout<<"\nx + y + w = "<<setprecision(9)<<x+y+w<<endl;
 cout<<"\nx * y = "<<setprecision(8)<<x*y<<endl;
 return 0;
}
```

Efekt działania tego programu dla zmiennej w równej 2,5 jest następujący:

podaj liczbę rzeczywistą: 2.5

x = 22.1111

y = 000003.55556

y = 0003.5555556

*w = **********2.5*

x + y + w = 28.1666667

*x * y = 78.617284*

Zadanie 3.1. Na podstawie kodu programu podanego w przykładzie 3.4 i wyników jego uruchomienia określ, w jaki sposób ustawiana jest precyzja dla liczb rzeczywistych za pomocą manipulatora `setprecision()`.

Podaj, z jaką precyzją wypisywane są wartości zmiennych rzeczywistych bez zastosowania manipulatora precyzji.

3.2.3. Zmienne, stałe, wskaźniki i referencje

Zmienne

Definicja

Zmienną nazywamy identyfikator, który może mieć przypisaną określoną wartość, którą podczas realizacji programu można zmieniać. Aby korzystać ze zmiennej, należy ją wcześniej zadeklarować, jednocześnie określając jej typ.

Deklaracja zmiennej ma postać:

```
typ identyfikator;
```

Nazwy zmiennych powinny być czytelne i ułatwiać piszącemu program dokonywanie w nim poprawek. Typy zmiennych zostały dokładnie omówione w podrozdziale 3.4., „Proste typy danych".

Przykład 3.5.

Przyjrzyjmy się przykładom **deklaracji zmiennych**:

```
int A;
float b, c, d;
char nazwa;
unsigned long int odleglosc;
string nazwaWydawnictwa, kolor_samochodu;
```

Podczas deklaracji zmiennych możliwe jest nadawanie im wartości początkowych przez **inicjalizację zmiennych**:

```
double liczba = 0.00125;
short int suma = 0, iloczyn = 1;
char znak = 'a';
```

Stałe

Definicja

Stałą nazywamy identyfikator o przypisanej określonej wartości, której podczas realizacji programu nie można zmieniać. Aby korzystać ze stałej, należy ją wcześniej zadeklarować, jednocześnie określając jej typ.

Deklaracja stałej ma postać:

```
const typ identyfikator = wartość;
```

Nazwy stałych, podobnie jak zmiennych, powinny być czytelne i ułatwiać piszącemu program dokonywanie w nim poprawek.

Przykład 3.6.

Przyjrzyjmy się przykładom deklaracji stałych:

```
const int B = 10;
const double a = 2.34, b = 3.125;
const char znak = 'W';
```

Wskaźniki

Definicja

Wskaźniki to adresy danych, struktur danych, obiektów klas lub funkcji. Wskaźnik zawiera informację o adresie oraz typie wskazywanego elementu w pamięci komputera.

Za pomocą operatora * możemy deklarować zmienne wskaźnikowe:

```
typ *identyfikator;
```

Wówczas `*identyfikator` to wartość zmiennej wskaźnikowej (czyli dynamicznej), a `identyfikator` to jej adres.

W języku C++ do dynamicznego przydziału i zwalniania pamięci stosuje się operatory **`new`** i **`delete`**. Operator **`new`** służy do przydzielania pamięci i ma składnię:

```
identyfikator = new typ_obiektu;
```

Wartością wyrażenia jest tutaj wskaźnik do utworzonego obiektu.

Operator **`delete`** wykorzystywany jest do zwalniania pamięci i ma następującą składnię:

```
delete identyfikator;
```

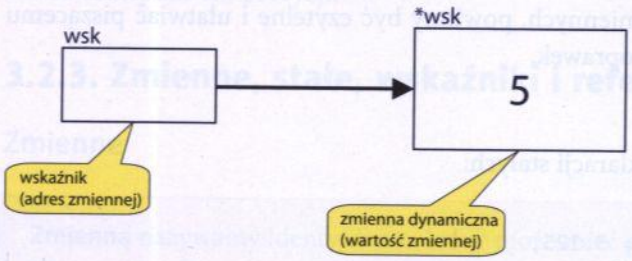
Operator **`delete`** usuwa obiekt, który jest wskazywany przez `identyfikator`.

Przykład 3.7.

Poniżej przedstawiono przykład dynamicznej alokacji pamięci dla zmiennej typu **int**:

```
int *wsk;
wsk=new int;
//Operacje wykonywane na zmiennej *wsk, na przykład: *wsk=5;
delete wsk;
```

Na rysunku 3.1 pokazano efekt wykonania operacji **new**. Zmienna **wsk** zawiera adres wskazujący na zmienną dynamiczną ***wsk**, której następnie przypisywana jest wartość 5.



Rysunek 3.1. Powiązanie wskaźnika ze zmienną dynamiczną

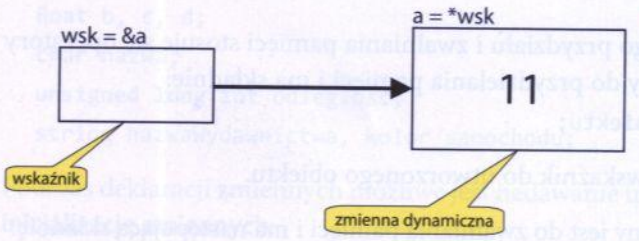
Do wyznaczenia wskaźnika zmiennej zadeklarowanej statycznie używany jest operator adresu **&**. Wykorzystuje się go do nadawania wartości początkowych zmiennym wskaźnikowym, a następnie dokonywania na nich zmian.

Przykład 3.8.

Przeanalizujmy przykład zastosowania operatora adresu:

```
int a;
int *wsk;
wsk=&a;
*wsk=11;
```

Na rysunku 3.2 przedstawiono efekt wykonania tych operacji.



Rysunek 3.2. Dwie nazwy dla jednej zmiennej dynamicznej

Wskaźnik `wsk` zawiera adres zmiennej dynamicznej `a`. Powoduje to, że zmienne `*wsk` i `a` są równoważne. Zmieniając wartość zmiennej dynamicznej `*wsk`, powodujemy zmiany zmiennej `a`, i odwrotnie. Przyczyna jest prosta — obydwie zmienne mają ten sam adres, a więc przeznaczone dla nich miejsce w pamięci komputera ma dwie nazwy.

Referencje

Definicja

Referencje, oznaczane znakiem `&`, zastępują zmienne, struktury danych lub obiekty klas. Operacje realizowane na referencji wykonywane są na reprezentowanej przez nią zmiennej, strukturze czy obiekcie.

Referencja zawsze musi mieć wartość początkową, która wiąże ją z reprezentowaną zmienną, strukturą lub obiektem:

```
typ &identyfikator = zmienna;
```

Przykład 3.9.

Przyjrzyjmy się przykładowi zastosowania referencji:

```
int a, b;
int &ref = a;
ref = 10;
b = ref;
```

Po wykonaniu powyższych instrukcji zmienna `a` reprezentowana przez referencję `ref` ma wartość `10`. Zmienna `b`, będąca zmienną, która nie jest powiązana z referencją `ref`, również ma wartość `10`. Operacje wykonywane na referencji `ref` i zmiennej `a` są równoważne.

3.2.4. Wyrażenia arytmetyczne, relacje i operatory logiczne

Operator przypisania (podstawienia)

Operator przypisania (patrz tabela 3.8) realizuje nadanie wartości zmiennej. Z lewej strony tego operatora musi być określona zmienna, z prawej — wyrażenie, którego wartość przypisujemy tej zmiennej.

Tabela 3.8. Operator przypisania

Operator	Działanie
=	Przypisanie wartości zmiennej

Przykład 3.10.

Przeanalizujmy przykłady zastosowania operatora przypisania:

```
a=4;

b=a+2;
```

`a=(b=8)-3;` Zagnieżdżenie instrukcji przypisania, równoważne instrukcjom `b=8; a=b-3.`

`a=b=c=0;` Wielokrotna instrukcja przypisania, równoważna instrukcjom `c=0; b=c; a=b.`

Operatory arytmetyczne

Operatory arytmetyczne (patrz tabela 3.9) to operatory dwuargumentowe realizujące operacje arytmetyczne.

Tabela 3.9. Zestawienie operatorów arytmetycznych

Operator	Działanie
*	Mnożenie
/	Dzielenie
+	Dodawanie
-	Odejmowanie
%	Reszta z dzielenia

Operator dzielenia ma dwa zastosowania, które zależą od typu argumentów. Jeśli obydwa argumenty są całkowite, wynik dzielenia również jest liczbą całkowitą. W tym przypadku uzyskujemy część całkowitą wyniku z dzielenia tych liczb. W sytuacji gdy chociaż jeden z argumentów jest rzeczywisty, wynikiem jest również liczba rzeczywista.

Przykład 3.11.

Przeanalizuj przedstawione w tabeli 3.10 przykłady dzielenia liczb całkowitych i rzeczywistych zapisane w języku C++.

Tabela 3.10. Przykłady wyrażeń zapisanych w języku C++ wykorzystujących operator dzielenia

Wyrażenie	Wartość wyrażenia
`9.0/2.0`	*4.5*
`9/2.0`	*4.5*
`9.0/2`	*4.5*
`9/2`	*4*

Tylko w jednym przypadku uzyskujemy wynik będący liczbą całkowitą. Aby wynik był liczbą całkowitą, obydwa argumenty muszą być wartościami całkowitymi.

Jeśli argumenty wykonywanej operacji dzielenia są zmiennymi całkowitymi, a potrzebny jest wynik rzeczywisty, pojawia się problem. Aby go rozwiązać, należy wymusić, by wynik dzielenia był rzeczywisty. Stosujemy wówczas tak zwane **rzutowanie**, które pokazuje poniżej podany przykład:

```
int a=5, b=2;
cout<<a/b<<"\t"<<(double)a/b<<endl;
```

Po wykonaniu tego kodu zostaną wypisane następujące wartości:

```
2    2.5
```

Operatory zmniejszania i zwiększania

Wyróżniamy dwa rodzaje operatorów zmniejszania i zwiększania (patrz tabela 3.11):

- prefiksowe — `++a`, `--a`,
- postfiksowe — `a++`, `a--`.

Tabela 3.11. Zestawienie operatorów zmniejszania i zwiększania

Operator	Działanie
`++`	Zwiększanie argumentu o 1 (inkrementacja)
`--`	Zmniejszanie argumentu o 1 (dekrementacja)

Jeśli operator występuje wewnątrz wyrażenia, to przy zapisie prefiksowym zwiększanie lub zmniejszanie argumentu jest wykonywane przed obliczeniem wartości wyrażenia, natomiast przy zapisie postfiksowym — po obliczeniu wartości wyrażenia. W tabeli 3.12 podano przykłady zastosowań tych operatorów.

Tabela 3.12. Przykłady zastosowań operatorów zmniejszania i zwiększania w zapisie postfiksowym i prefiksowym

Przykład zastosowania	Wartości zmiennych po wykonaniu operacji

`int a=3, b=4, c;`
`++a;`
`b--;`
`c = a + b;`
	a = 4
	b = 3
	c = 7

`int a=3, b=4, c;` `c = ++a + b;`	a = 4 b = 4 c = 8
`int a=3, b=4, c;` `c = a + b--;`	a = 3 b = 3 c = 7
`int a=3, b=4, c;` `c = ++a + b--;`	a = 4 b = 3 c = 8