

12

Tworzenie i inicjowanie obiektów

12.1. Konstruktory

Konstruktor (ang. *constructor*) jest specjalną funkcją członkowską klasy (ang. *class member function*). Konstruktor jest wywoływany automatycznie w sytuacji, w której jest tworzony nowy obiekt należący do tej klasy — czyli obiekt będący jej instancją. Oprócz tego konstruktory są wykorzystywane do inicjalizacji (nadania wartości początkowych) zmiennym członkowskim (ang. *member variables*) tworzonego obiektu, a czasem również w celu wykonania innych działań pomocniczych.

UWAGA

W języku C++ konstruktory tak naprawdę nie tworzą nowych obiektów. To kompilator odpowiada za utworzenie obiektu, ponieważ to on rezerwuje (alokuje) pamięć operacyjną niezbędną do zapamiętania obiektu — przed faktycznym wywołaniem konstruktora. Obiekt należący do danej klasy zostanie utworzony tylko wtedy, gdy kompilator: (1) albo znajdzie definicję konstruktora pasującego do wyrażenia zawierającego wywołanie konstruktora (w sposób jawny lub niejawny), albo (2) sam utworzy niejawny konstruktor domyślny.

W obrębie (we wnętrzu) klasy może być zawarta zarówno kompletna definicja konstruktora, jak i wyłącznie jego prototyp (deklaracja). W tym drugim przypadku definicję konstruktora zamieszczoną na zewnątrz klasy — podobnie jak definicję zwykłej funkcji członkowskiej (metody) — należy poprzedzić nazwą klasy oraz operatorem zakresu `::`. Klasa może mieć wiele konstruktorów — w zależności od potrzeb.

Konstruktory mają pewne szczególne właściwości, które odróżniają je od zwykłych funkcji członkowskich klasy, mianowicie:

- nazwa konstruktora jest taka sama jak nazwa klasy,
- konstruktor nie zwraca do swojego otoczenia żadnej/żadnych wartości,
- każdy konstruktor jest metodą publiczną.

Biorąc pod uwagę pierwszą z wymienionych powyżej właściwości konstruktorów, identyfikator konstruktora odpowiada jednoznacznie identyfikatorowi klasy — są one identyczne. Na przykład, jeśli `Pracownik` jest nazwą klasy, to identyfikatorem konstruktora również jest `Pracownik`. Dotyczy to wszystkich konstruktorów zdefiniowanych w klasie.

Konstruktor, jako integralna funkcja członkowska klasy, ma jednoznacznie określony interfejs, który pozwala mu komunikować się z otoczeniem. Z drugiej z wymienionych wcześniej charakterystycznych właściwości konstruktorów wynika, że dla konstruktora można określić jedynie wejście. Tym samym konstruktor nie zwraca na zewnątrz żadnej wartości. Stąd jego definicja nie zawiera typu zwracanej wartości. Konstruktor nie ma też żadnych parametrów/argumentów wyjściowych.

Wejście konstruktora stanowią parametry wejściowe, które odpowiadają wszystkim lub wybranym zmiennym członkowskim klasy. Lista parametrów konstruktora może być również pusta. Typy parametrów konstruktora powinny być zgodne z typami zmiennych członkowskich klasy reprezentowanych w jego definicji/deklaracji.

Trzecia z wymienionych wcześniej właściwości konstruktorów oznacza, że są dostępne wszędzie — w całym otoczeniu klasy, w której zostały zdefiniowane, oczywiście z uwzględnieniem zakresu widoczności tej klasy.

12.1.1. Konstruktory domyślne

W ogólności w języku C++ obowiązuje zasada, że **konstruktor domyślny** (ang. *default constructor*) jest:

- albo konstruktorem bezparametrowym (ang. *unparametrized/non parametrized constructor*),
- albo konstruktorem, w którym wszystkie parametry mają przypisane wartości domyślne.

Konstruktor niejawny

Klasa zdefiniowana przez programistę może zawierać jawną definicję konstruktora lub nie. Jeśli w danej klasie żaden konstruktor nie został zdefiniowany w sposób jawny, kompilator automatycznie utworzy „po cichu” (niejawnie) konstruktor domyślny.

Konstruktor domyślny utworzony „po cichu” przez kompilator (a nie zdefiniowany jawnie przez programistę) często jest nazywany **konstruktorem niejawnym** (ang. *implicit*

constructor). Taki konstruktor nie ma żadnych parametrów. Jeżeli programista zdefiniuje w klasie jakiegokolwiek „własny” konstruktor, kompilator nie utworzy konstruktora niejawnego. Konstruktor niejawny (utworzony automatycznie przez kompilator) może być wywoływany w programie również w sposób niejawny. W praktyce można to zrealizować na dwa sposoby:

1. Bez inicjalizacji zmiennych członkowskich klasy.
2. Z inicjalizacją zmiennych członkowskich wartościami domyślnymi.

Oba wymienione powyżej sposoby zilustrowano, odpowiednio, w przykładzie 12.1 oraz przykładzie 12.2.

Przykład 12.1

```
#include <iostream>
using namespace std;

// Definicja struktury Data:
struct Data {
    // Deklaracje pól struktury:
    int dd, mm, rr;
};

// Definicja klasy Pracownik:
class Pracownik {
public:
    int id;
    string imie, nazwisko;
    Data data_urodzenia;
};

/* UWAGA
* Należy zwrócić uwagę na to, że wewnątrz klasy Pracownik nie jest zawarta definicja żadnego konstruktora.
* Ponadto żadna ze zmiennych członkowskich klasy Pracownik nie została zainicjowana.
*/

int main() {
    // Utworzenie obiektu pracownik jako instancji klasy Pracownik:
    Pracownik pracownik; // wywołanie niejawnego konstruktora klasy Pracownik w sposób niejawny
    /* UWAGA
    * Pomimo że w klasie Pracownik nie zdefiniowano żadnego konstruktora, to po utworzeniu obiektu pracownik —
    * czyli po zaalokowaniu dla niego niezbędnej pamięci operacyjnej przez kompilator —
    * następuje automatyczne wywołanie konstruktora domyślnego bezparametrowego,
    * czyli konstruktora niejawnego. Zmienne członkowskie obiektu nie zostały w tym przypadku zainicjowane!
    */
    // Wyświetlenie wartości zmiennych członkowskich obiektu pracownik:
```

```

cout << "Numer identyfikacyjny " << pracownik.id << endl;
cout << "Imię: " << pracownik.imie << endl;
cout << "Nazwisko: " << pracownik.nazwisko << endl;
cout << "Data urodzenia:" << endl;
cout << "dzień = " << pracownik.data_urodzenia.dd << endl;
cout << "miesiąc = " << pracownik.data_urodzenia.mm << endl;
cout << "rok = " << pracownik.data_urodzenia.rr << endl;

```

```

return 0;

```

```

}

```

W programie zdefiniowano klasę `Pracownik` zawierającą zmienne członkowskie: `id`, `imie`, `nazwisko` i `data_urodzenia`. Żadna z tych zmiennych nie została zainicjowana w obrębie definicji klasy.

Ponadto definicja klasy `Pracownik` nie zawiera definicji żadnego konstruktora. Tym samym kompilator utworzy „po cichu” (niejawnie) konstruktor domyślny bezparametrowy, czyli konstruktor niejawny. Po zaalokowaniu pamięci operacyjnej dla obiektu `pracownik` wspomniany konstruktor niejawny zostaje automatycznie wywołany również w sposób niejawny: `Pracownik pracownik;`.

Żadna ze zmiennych członkowskich klasy nie została zainicjowana. Wartości zmiennych członkowskich wyświetlone kontrolnie na ekranie monitora są przypadkowe.

Ćwiczenie 12.1

Zmodyfikuj program zawarty w przykładzie 12.1 — każdą ze zmiennych członkowskich klasy `Pracownik` zainicjuj w obrębie jej definicji, np. `int id = 1; string imie = "imie domyślne"`. Zinterpretuj uzyskane wyniki.

Przykład 12.2

```

#include <iostream>
using namespace std;

```

```

// Definicja struktury Data:

```

```

struct Data {
    int dd, mm, rr;
};

```

```

// Definicja klasy Pracownik:

```

```

class Pracownik {
public:
    int id {-1};
    string imie {"Imię domyślne"}, nazwisko {"Nazwisko domyślne"};
    Data data_urodzenia {31, 12, 1899};
};

```



```

/* UWAGA
 * Wewnątrz definicji klasy Pracownik nie jest zawarta definicja żadnego konstruktora.
 * Każda ze zmiennych członkowskich klasy Pracownik została zainicjowana ustaloną wartością domyślną.
 */

int main() {
    // Utworzenie obiektu pracownik jako instancji klasy Pracownik:
    Pracownik pracownik; // wywołanie konstruktora niejawnego również w sposób niejawny
    /* UWAGA
     * Po zarezerwowaniu przez kompilator pamięci operacyjnej dla zmiennej pracownik, czyli po
     * utworzeniu obiektu pracownik, następuje automatyczne wywołanie konstruktora niejawnego.
     * Zmienne członkowskie obiektu zostały zainicjowane wartościami domyślnymi określonymi w definicji klasy.
     */
    // Wyświetlenie wartości zmiennych członkowskich obiektu pracownik:
    cout << "Numer identyfikacyjny " << pracownik.id << endl;
    cout << "Imię: " << pracownik.imie << endl;
    cout << "Nazwisko: " << pracownik.nazwisko << endl;
    cout << "Data urodzenia:" << endl;
    cout << "dzień = " << pracownik.data_urodzenia.dd << endl;
    cout << "miesiąc = " << pracownik.data_urodzenia.mm << endl;
    cout << "rok = " << pracownik.data_urodzenia.rr << endl;

    return 0;
}

```

W programie zdefiniowano klasę Pracownik zawierającą zmienne członkowskie: id, imie, nazwisko i data_urodzenia, które w obrębie tej definicji zostały zainicjowane ustalonymi wartościami domyślnymi.

Definicja klasy Pracownik nie zawiera definicji żadnego konstruktora. Dlatego też kompilator utworzy w sposób automatyczny konstruktor niejawny (czyli konstruktor domyślny bezparametrowy).

Po utworzeniu obiektu pracownik, czyli po zaalokowaniu pamięci operacyjnej niezbędnej do zapamiętania zmiennej pracownik, zostaje automatycznie wywołany konstruktor niejawny: Pracownik pracownik;. Wywołanie konstruktora niejawnego również w sposób niejawny skutkuje zainicjowaniem wszystkich jego zmiennych członkowskich wartościami domyślnymi określonymi w definicji klasy.

Ćwiczenie 12.2

Zmodyfikuj program zawarty w przykładzie 12.2 — zainicjuj zmienne członkowskie klasy Pracownik przy użyciu innego sposobu inicjalizacji, np. inicjalizacji kopiującej.

Konstruktor domyślny zdefiniowany przez programistę

Konstruktor domyślny może być również zdefiniowany samodzielnie przez programistę. Należy jednak pamiętać o tym, że konstruktor domyślny jest z założenia konstruktorem bezparametrowym.

Jeżeli w klasie zdefiniowano co najmniej jeden konstruktor, kompilator nie utworzy konstruktora niejawnego. Innymi słowy, kompilator nie utworzy konstruktora niejawnego, jeśli programista zdefiniował w klasie chociaż jeden własny konstruktor — np. konstruktor domyślny.

Przykład 12.3

```
#include <iostream>
using namespace std;

// Definicja struktury Data:
struct Data {
    int dd, mm, rr;
};

// Definicja klasy Pracownik:
class Pracownik {
public:
    int id;
    string imie, nazwisko;
    Data data_urodzenia;
    // Definicja konstruktora domyślnego:
    Pracownik() {
        id = -1;
        imie = "Imię domyślne", nazwisko = "Nazwisko domyślne";
        data_urodzenia = {31, 12, 1899};
    }
};

/* UWAGA
 * W konstruktorze domyślnym zdefiniowanym samodzielnie przez programistę zainicjowano zadanymi wartościami
 * domyślnymi wszystkie zmienne członkowskie klasy Pracownik. Nie jest to jednak wymagane. W ogólności
 * ciało konstruktora domyślnego może być puste.
 */

int main() {
    // Utworzenie obiektu pracownik jako instancji klasy Pracownik:
    Pracownik pracownik;
    /* UWAGA
     * Po utworzeniu obiektu pracownik przez kompilator następuje automatyczne wywołanie konstruktora
     * domyślnego zdefiniowanego samodzielnie przez programistę. Wszystkie zmienne członkowskie obiektu
```


*pracownik zostały zainicjowane wartościami domyślnymi określonymi w definicji tego konstruktora.

*/

// Wyświetlenie wartości zmiennych członkowskich obiektu pracownik:

```
cout << "Numer identyfikacyjny " << pracownik.id << endl;
cout << "Imię: " << pracownik.imie << endl;
cout << "Nazwisko: " << pracownik.nazwisko << endl;
cout << "Data urodzenia:" << endl;
cout << "dzień = " << pracownik.data_urodzenia.dd << endl;
cout << "miesiąc = " << pracownik.data_urodzenia.mm << endl;
cout << "rok = " << pracownik.data_urodzenia.rr << endl;
```

```
return 0;
```

```
}
```

W programie zdefiniowano klasę Pracownik ze zmiennymi członkowskimi: id, imię, nazwisko i data_urodzenia. Definicja klasy Pracownik zdefiniowanej tutaj — w odróżnieniu od klasy Pracownik z przykładu 12.1 oraz przykładu 12.2 — obejmuje definicję konstruktora domyślnego. W ciele wspomnianego konstruktora zainicjowano wszystkie zmienne członkowskie klasy zadanymi wartościami domyślnymi.

Po zaalokowaniu przez kompilator pamięci operacyjnej dla obiektu pracownik następuje niejawne wywołanie konstruktora domyślnego zdefiniowanego samodzielnie przez programistę: Pracownik pracownik;. Skutkuje to zainicjowaniem wszystkich zmiennych członkowskich obiektu pracownik wartościami domyślnymi określonymi w definicji tego konstruktora.

Ćwiczenie 12.3

Zmodyfikuj kod źródłowy programu zawartego w przykładzie 12.3 — zainicjuj zmienne członkowskie klasy w definicji konstruktora domyślnego w inny sposób, niż zaproponowano w podanym przykładzie.

12.1.2. Konstruktory z parametrami

Konstruktory z parametrami, inaczej **konstruktory parametryczne** (ang. *parametrized constructors*), to konstruktory, które mają parametry. Konstruktory parametryczne są definiowane samodzielnie przez programistę. Parametry te odpowiadają zmiennym członkowskim klasy: wszystkim lub wybranym. Tym samym za pomocą argumentów wywołania konstruktora parametrycznego można zainicjować określonymi wartościami wszystkie lub wybrane zmienne członkowskie klasy. Wspomniane wartości inicjujące są zadane (ustalone) na poziomie funkcji wywołującej konstruktor, np. funkcji main().

Jeżeli liczba parametrów konstruktora parametrycznego nie obejmuje wszystkich zmiennych członkowskich klasy, wywołanie tego konstruktora z zadanymi argumentami powoduje, że pozostałe zmienne członkowskie (czyli te, które nie zostały zainicjowane za pośrednictwem argumentów):

- albo nie zostają zainicjowane,
- albo zostają zainicjowane wartościami domyślnymi.

Pierwszy z wymienionych powyżej przypadków dotyczy sytuacji, w której rozpatrywanym zmiennym członkowskim — tym, których nie obejmują argumenty wywołania konstruktora — nie przypisano wartości domyślnych w obrębie definicji klasy.

Drugi przypadek obejmuje sytuację przeciwną, czyli zmiennym członkowskim zostały w definicji klasy przypisane wartości domyślne.

Rozpatrując równocześnie obydwa omówione przypadki, należy uwzględnić to, że kompilator, o czym już wspomniano, nie utworzy konstruktora domyślnego po samodzielnym zdefiniowaniu konstruktora przez programistę.

Przykład 12.4

```
#include <iostream>
using namespace std;

// Definicja struktury Data:
struct Data {
    // Deklaracje pól struktury Data:
    int dd, mm, rr;
};

// Definicja klasy Pracownik:
class Pracownik {
public:
    // Deklaracje zmiennych członkowskich:
    int id;
    string imie, nazwisko;
    Data data_urodzenia;

    // Prototyp konstruktora parametrycznego:
    Pracownik(string, string, Data);
};

// Definicja konstruktora parametrycznego:
Pracownik::Pracownik(string pImie, string pNazwisko, Data pDataUrodzenia) {
    imie = pImie;
    nazwisko = pNazwisko;
    data_urodzenia = pDataUrodzenia;
}

// PROGRAM GŁÓWNY
int main() {
    // Niejawne wywołanie konstruktora parametrycznego z zadanymi argumentami:
    Pracownik pracownik("Jan", "Kowalski", {31, 12, 2000});
```

/ UWAGA*

- * Po zaalokowaniu pamięci operacyjnej dla obiektu pracownik będącego instancją klasy Pracownik,*
- * jego zmienne członkowskie imie, nazwisko i data_urodzenia zostają zainicjowane wartościami argumentów*
- * podanych w wywołaniu konstruktora parametrycznego.*
- * Jednocześnie zmienna członkowska id nie została zainicjowana!*

**/*

// Wyświetlenie wartości przechowywanych w zmiennych członkowskich obiektu pracownik:

```
cout << "Numer identyfikacyjny: " << pracownik.id << endl;
cout << "Imię: " << pracownik.imie << endl;
cout << "Nazwisko: " << pracownik.nazwisko << endl;
cout << "Data urodzenia:" << endl;
cout << "dzień = " << pracownik.data_urodzenia.dd << endl;
cout << "miesiąc = " << pracownik.data_urodzenia.mm << endl;
cout << "rok = " << pracownik.data_urodzenia.rr << endl;
```

```
return 0;
```

W programie zdefiniowano klasę Pracownik ze zmiennymi członkowskimi: id, imie, nazwisko i data_urodzenia. Definicja klasy Pracownik zdefiniowanej tutaj obejmuje definicję konstruktora parametrycznego — w odróżnieniu od klasy Pracownik z przykładu 12.3, w której zdefiniowano konstruktor domyślny.

Zdefiniowany konstruktor parametryczny ma dwa zadania. Po pierwsze kompilator, natrafwszy w kodzie źródłowym programu na wyrażenie Pracownik pracownik("Jan", "Kowalski", {31, 12, 2000});, będzie poszukiwał w klasie Pracownik pasującej do niego definicji konstruktora parametrycznego. Po znalezieniu definicji odpowiedniego konstruktora obiekt pracownik zostanie utworzony, a następnie wspomniany konstruktor jest wywoływany.

Drugim zadaniem omawianego konstruktora jest inicjalizacja zmiennych członkowskich imie, nazwisko i data_urodzenia obiektu pracownik wartościami odpowiadających im argumentów wywołania konstruktora. Należy zwrócić uwagę, że zdefiniowany w klasie Pracownik (i wywołany w programie głównym) konstruktor parametryczny nie pozwala na inicjalizację zmiennej członkowskiej id. Wartość tej zmiennej wyświetlana na ekranie monitora jest przypadkowa.

Konstruktor parametryczny jest wywoływany w programie głównym w sposób niejawni. Wyrażenie, w którym konstruktor ten jest wywoływany jawnie, może mieć postać: Pracownik pracownik = Pracownik("Jan", "Kowalski", {31, 12, 2000});.

UWAGA

Dobłą praktyką programistyczną jest to, aby podczas tworzenia obiektu inicjować jego wszystkie zmienne członkowskie. Tym samym konstruktor parametryczny powinien pozwalać na inicjalizację wszystkich zmiennych członkowskich klasy.

Ćwiczenie 12.4

Zmodyfikuj program zawarty w przykładzie 12.4 — zainicjuj zmienną członkowską `id` w definicji klasy `Pracownik` wartością 1. Zinterpretuj uzyskany wynik. W następnym teście zmodyfikuj definicję (i prototyp) konstruktora parametrycznego w taki sposób, aby jego wywołanie pozwalało na inicjalizację wszystkich zmiennych członkowskich klasy `Pracownik`. Wywołaj ten konstruktor, a wartość argumentu odpowiadającego zmiennej członkowskiej `id` ustal na 10. Zinterpretuj uzyskany wynik.

12.1.3. Przeciążanie konstruktorów

Konstruktory są funkcjami specjalnymi, które można przeciążać tak samo jak inne funkcje. Dlatego też w definicji klasy mogą być zawarte definicje wielu konstruktorów, które różnią się od siebie liczbą i/lub typem parametrów.

Przeciążanie konstruktorów (ang. *overloading constructors*) jest realizowane w praktyce według tych samych zasad, co przeciążanie zwykłych funkcji.

UWAGA

Przeciążanie funkcji (ang. *function overloading*) zostało szczegółowo omówione w podrödziale 8.5.

Dla przypomnienia — przeciążanie funkcji polega na tym, że w tym samym zakresie (ang. *scope*) definiowane są funkcje, które mają taką samą nazwę, ale różnią się od siebie liczbą i/lub typem parametrów. W przypadku przeciążania konstruktorów wspomniany zakres jest określony przez **zakres klasy** (ang. *class scope*), który obejmuje jej ciało (ang. *class body*).

Kompilator, natrafiwszy w kodzie źródłowym programu na wyrażenie zawierające wywołanie konstruktora, automatycznie wybiera — na podstawie wyniku analizy liczby i typów argumentów tego wywołania — jego odpowiednią wersję zdefiniowaną w klasie.

Przykład 12.5

```
#include <iostream>
using namespace std;

// Definicja struktury Data:
struct Data {
    int dd, mm, rr;
};

// Definicja klasy Pracownik:
class Pracownik {
public:
    // Deklaracje zmiennych członkowskich:
```

```

int id;
string imie, nazwisko;
// Deklaracja zmiennej członkowskiej data_urodzenia połączona z inicjalizacją jej elementów składowych
// wartościami domyślnymi:
Data data_urodzenia {31, 12, 1899};

// Prototyp konstruktora domyślnego:
Pracownik();
// Prototypy konstruktorów parametrycznych:
Pracownik(int, string, string);
Pracownik(int, string, string, Data);
// Prototyp publicznej metody dostępowej:
void wyswietlDane();

};
// Definicja konstruktora domyślnego:
Pracownik::Pracownik() {
    id = -1;
    imie = "Imię domyślne";
    nazwisko = "Nazwisko domyślne";
    data_urodzenia = {1, 1, 1900};
}
// Definicje konstruktorów parametrycznych:
Pracownik::Pracownik(int pId, string pImie, string pNazwisko) {
    id = pId;
    imie = pImie;
    nazwisko = pNazwisko;
}
Pracownik::Pracownik(int pId, string pImie, string pNazwisko,
    Data pDataUrodzenia) {
    id = pId;
    imie = pImie;
    nazwisko = pNazwisko;
    data_urodzenia = pDataUrodzenia;
}
// Definicja metody wyswietlDane()
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
    cout << "Data urodzenia:" << endl;
    cout << "dzień = " << data_urodzenia.dd << endl;
    cout << "miesiąc = " << data_urodzenia.mm << endl;
    cout << "rok = " << data_urodzenia.rr << endl;
}

```



```
// PROGRAM GŁÓWNY
int main() {
    // Utworzenie obiektu pracownik1 będącego instancją klasy Pracownik:
    Pracownik pracownik1;
    /* UWAGA
    * Po zaalokowaniu pamięci dla obiektu pracownik1 został wywołany w sposób niejawni konstruktor domyślny
    * zdefiniowany przez programistę. Zmienne członkowskie tego obiektu zostały zainicjowane wartościami
    * domyślnymi określonymi w definicji tego konstruktora.
    */
    // Wyświetlenie wartości przechowywanych w zmiennych członkowskich obiektu pracownik1:
    pracownik1.wyswietlDane();
    cout << endl;

    // Utworzenie obiektu pracownik2 jako instancji klasy Pracownik:
    Pracownik pracownik2(1, "Jan", "Kowalski");
    /* UWAGA
    * Po zarezerwowaniu pamięci operacyjnej dla obiektu pracownik2 został wywołany w sposób niejawni
    * konstruktor parametryczny zdefiniowany przez programistę. Zmienne członkowskie id, imie i nazwisko obiektu
    * zostały zainicjowane zadanymi wartościami argumentów wywołania tego konstruktora.
    * Zmienna członkowska data_urodzenia została zainicjowana wartościami domyślnymi jej składowych, które
    * zostały ustalone w obrębie definicji klasy.
    */
    pracownik2.wyswietlDane();
    cout << endl;

    // Utworzenie obiektu pracownik3 jako instancji klasy Pracownik:
    Pracownik pracownik3(10, "Adam", "Nowak", {10, 10, 2000});
    /* UWAGA
    * Konstruktor parametryczny został wywołany w sposób niejawni. Wszystkie zmienne członkowskie obiektu
    * pracownik3 zostały zainicjowane zadanymi wartościami argumentów wywołania tego konstruktora.
    */
    pracownik3.wyswietlDane();

    return 0;
}
```

W klasie Pracownik zdefiniowano trzy konstruktory. Pierwszy z nich jest konstruktorem domyślnym, ponieważ nie ma żadnych parametrów. Dwa pozostałe to konstruktory parametryczne. Pierwszy z nich pozwala inicjować wybrane zmienne członkowskie obiektu klasy Pracownik, a drugi — wybrane.

Wszystkie z wymienionych konstruktorów zostały zdefiniowane w tym samym zakresie — zakresie klasy Pracownik. Konstruktory te mają oczywiście tę samą nazwę, ale jednocześnie różnią się od siebie liczbą i typem parametrów. Mamy więc tutaj do czynienia z mechanizmem przeciążania konstruktorów.

Konstruktory są wywoływane w programie głównym. Kompilator każdorazowo sprawdza liczbę i typy argumentów tych wywołań i dopasowuje automatycznie odpowiedni — zdefiniowany w klasie — konstruktor w celu jego wywołania.

Ćwiczenie 12.5

Zmodyfikuj program z przykładu 12.5 — przydziel wartości domyślne wszystkim zmiennym członkowskim klasy `Pracownik` w obrębie jej definicji. Wykonaj testy zmodyfikowanego programu. Zinterpretuj uzyskane rezultaty.

12.2. Inicjalizacja obiektów

Inicjalizacja obiektu (ang. *object initialization*) polega na nadaniu wartości początkowych jego zmiennym członkowskim. Różne obiekty były inicjowane w wielu przykładowych programach zaprezentowanych wcześniej w tym rozdziale. Jednakże zwracaliśmy tam uwagę na aspekty praktyczne stosowania konstruktorów, nie skupiając się na zagadnieniu ich inicjalizacji. W niniejszym podrozdziale tematyka inicjalizacji obiektów została usystematyzowana.

W przypadku ogólnym obiekty można inicjować wartościami domyślnymi lub wartościami zadanymi.

12.2.1. Inicjalizacja obiektów wartościami domyślnymi

Obiekty można inicjować wartościami domyślnymi na trzy sposoby:

- z użyciem **inicjalizatorów wewnątrzklasowych** (ang. *in-class initializers*) bezpośrednio w definicji (wewnątrz) klasy,
- za pomocą zdefiniowanego konstruktora domyślnego (ang. *default constructor*) zawierającego w swoim ciele inicjalizatory zmiennych członkowskich,
- z zastosowaniem konstruktora parametrycznego (ang. *parametrized constructor*) z parametrami domyślnymi — jeśli nie zdefiniowano konstruktora domyślnego.

Należy zaznaczyć, że inicjalizacja z użyciem konstruktora (domyślnego i parametrycznego) ma wyższy priorytet niż inicjalizacja za pomocą inicjalizatorów wewnątrzklasowych.

Przykład 12.6

```
#include <iostream>
using namespace std;

// Definicja klasy Pracownik:
class Pracownik {
public:
    // Definicje zmiennych członkowskich:
    int id = -1; // inicjalizator wewnątrzklasowy
```



```

string imie = "Imię domyślne"; // inicjalizator wewnątrzklasowy
string nazwisko = "Nazwisko domyślne"; // inicjalizator wewnątrzklasowy
// Prototyp konstruktora domyślnego:
Pracownik();
// Prototyp konstruktora parametrycznego:
Pracownik(int, string, string);
// Prototyp funkcji członkowskiej:
void wyswietlDane();

};
// Definicje konstruktorów:
Pracownik::Pracownik() {
    cout << "Komunikat kontrolny: wywołanie konstruktora domyślnego ..."
        << endl;
}
Pracownik::Pracownik(int pId, string pImie, string pNazwisko) {
    id = pId;
    imie = pImie;
    nazwisko = pNazwisko;
}
// Definicja funkcji członkowskiej wyswietlDane():
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
}

int main() {
    // Utworzenie obiektu pracownik:
    Pracownik pracownik;
    /* Uwaga
    * Konstruktor domyślny został wywołany w sposób niejawni po przydzieleniu pamięci operacyjnej dla obiektu
    * pracownik. Zmienne członkowskie obiektu pracownik zostały zainicjowane wartościami domyślnymi
    * określonymi w definicji klasy, czyli za pomocą inicjalizatorów wewnątrzklasowych.
    */
    pracownik.wyswietlDane();

    return 0;
}

```

W programie zdefiniowano klasę `Pracownik`, w której zawarto wewnątrzklasowe inicjalizatory wszystkich zmiennych członkowskich.

Klasa `Pracownik` ma dwa konstruktory — domyślny i parametryczny. Konstruktor domyślny zdefiniowany przez programistę nie zawiera inicjalizatorów zmiennych członkowskich.

Konstruktor domyślny został wywołany w programie w sposób niejawni: Pracownik pracownik; Zmienne członkowskie id, imię i nazwisko zostały zainicjowane za pomocą inicjalizatorów wewnątrzklasowych zawartych w definicji klasy Pracownik.

Ćwiczenie 12.6

Zmodyfikuj program zawarty w przykładzie 12.6 — w klasie Pracownik zdefiniuj dodatkową zmienną członkowską o nazwie data_zatrudnienia należącą do typu strukturalnego Data. Struktura Data powinna zawierać trzy pola umożliwiające zapamiętanie zadanej daty — dnia, miesiąca i roku. Zainicjuj obiekt pracownik ustalonymi wartościami domyślnymi z użyciem inicjalizatorów wewnątrzklasowych.

Przykład 12.7

```
#include <iostream>
using namespace std;

// Definicja klasy Pracownik:
class Pracownik {
public:
    // Definicje zmiennych członkowskich:
    int id = -1; // wewnątrzklasowy inicjalizator zmiennej członkowskiej
    string imię = "Imię domyślne"; // wewnątrzklasowy inicjalizator zmiennej członkowskiej
    string nazwisko = "Nazwisko domyślne"; // wewnątrzklasowy inicjalizator zmiennej
                                         // członkowskiej

    // Prototyp konstruktora domyślnego:
    Pracownik();
    // Prototyp konstruktora parametrycznego:
    Pracownik(int, string, string);
    // Prototyp funkcji członkowskiej:
    void wyświetlDane();
};

// Definicje konstruktorów:
Pracownik::Pracownik() {
    id = 0; // inicjalizacja zmiennej członkowskiej wartością domyślną
    imię = "I m i e"; // inicjalizacja zmiennej członkowskiej wartością domyślną
    nazwisko = "N a z w i s k o"; // inicjalizacja zmiennej członkowskiej wartością domyślną

    cout << "Komunikat kontrolny: wywołanie konstruktora domyślnego ..."
         << endl;
}

Pracownik::Pracownik(int pId, string pImię, string pNazwisko) {
    id = pId;
    imię = pImię;
    nazwisko = pNazwisko;
}
```



```
// Definicja funkcji członkowskiej:
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
}

int main() {
    // Utworzenie obiektu pracownik1:
    Pracownik pracownik1;
    /* Uwaga
    * Konstruktor domyślny został wywołany w sposób niejawni. Zmienne członkowskie obiektu pracownik1
    * zostały zainicjowane wartościami domyślnymi ustalonymi w definicji tego konstruktora.
    */
    pracownik1.wyswietlDane();

    // Utworzenie obiektu pracownik2:
    Pracownik pracownik2 = Pracownik();
    /* Uwaga
    * Konstruktor domyślny został wywołany w sposób jawny. Zmienne członkowskie obiektu pracownik2
    * zostały zainicjowane wartościami domyślnymi określonymi w definicji tego konstruktora.
    */
    pracownik2.wyswietlDane();

    return 0;
}
```

W programie zdefiniowano klasę `Pracownik`, w której wszystkie zmienne członkowskie zainicjowano wartościami domyślnymi z wykorzystaniem inicjalizatorów wewnątrzklasowych.

Należy zwrócić uwagę, że wartości domyślne zmiennych członkowskich klasy zostały również określone w ciele konstruktora domyślnego. Jednakże wartości te są odmienne od wartości domyślnych ustalonych za pomocą inicjalizatorów wewnątrzklasowych.

Obiekt `pracownik1` został utworzony przez niejawne wywołanie konstruktora domyślnego, a obiekt `pracownik2` — jawne wywołanie tego samego konstruktora. W obu przypadkach zmienne członkowskie `id`, `imie` i `nazwisko` zostały zainicjowane z zastosowaniem inicjalizatorów zawartych w definicji (ciele) konstruktora domyślnego, który ma wyższy priorytet od inicjalizatorów wewnątrzklasowych.

Ćwiczenie 12.7

Zmodyfikuj program zawarty w przykładzie 12.7 — w klasie `Pracownik` zdefiniuj dodatkową zmienną członkowską o nazwie `data_zatrudnienia` należącą do typu strukturalnego `Data`. Struktura `Data` powinna zawierać trzy pola umożliwiające zapamiętanie zadanej daty — dnia, miesiąca i roku. Zainicjuj obiekt `pracownik` należący do klasy `Pracownik` wartościami domyślnymi określonymi w definicji konstruktora domyślnego.

Przykład 12.8

```

#include <iostream>
using namespace std;

// Definicja klasy Pracownik:
class Pracownik {
public:
    // Deklaracje zmiennych członkowskich:
    int id = -1;
    string imie = "Imię domyślne";
    string nazwisko = "Nazwisko domyślne";
    // Prototyp konstruktora parametrycznego:
    pracownik(int pId = 0, string pImie = "I m i ę",
               string pNazwisko = "N a z w i s k o");
    // dla wszystkich parametrów powyżej określone zostały wartości domyślne

    // Prototyp funkcji członkowskiej:
    void wyswietlDane();
};

/* UWAGA
 * W klasie Pracownik nie zdefiniowano konstruktora domyślnego.
 */

// Definicja konstruktora parametrycznego:
Pracownik::Pracownik(int pId, string pImie, string pNazwisko) {
    id = pId;
    imie = pImie;
    nazwisko = pNazwisko;
}

// Definicja funkcji członkowskiej:
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
}

int main() {
    Pracownik pracownik = Pracownik();
    /* Uwaga
     * Konstruktor parametryczny został wywołany w sposób jawny — ale bez żadnych argumentów. Tym samym
     * zmienne członkowskie obiektu pracownik zostały zainicjowane wartościami domyślnymi jego parametrów.
     */
    pracownik.wyswietlDane();

    return 0;
}

```


W programie zdefiniowano klasę `Pracownik`, w której wszystkie zmienne członkowskie zainicjowano wartościami domyślnymi z użyciem inicjalizatorów wewnątrzklasowych. Wartości domyślne zmiennych członkowskich zostały również ustalone dla parametrów zdefiniowanego konstruktora parametrycznego.

Obiekt `pracownik` został utworzony przez jawne wywołanie konstruktora parametrycznego. Zmienne członkowskie `id`, `imie` i `nazwisko` zostały zainicjowane za pomocą wartości domyślnych stanowiących parametry domyślne konstruktora, które mają wyższy priorytet od inicjalizatorów wewnątrzklasowych.

Ćwiczenie 12.8

Zmodyfikuj program zawarty w przykładzie 12.8 — w klasie `Pracownik` zdefiniuj dodatkową zmienną członkowską o nazwie `data_zatrudnienia` należącą do typu strukturalnego `Data`. Struktura `Data` powinna zawierać trzy pola umożliwiające zapamiętanie zadanej daty — dnia, miesiąca i roku. Zainicjuj obiekt `pracownik` należący do klasy `Pracownik` wartościami domyślnymi określonymi jako parametry domyślne zdefiniowanego samodzielnie konstruktora parametrycznego.

12.2.2. Inicjalizacja obiektów wartościami zadanymi

W ogólności inicjalizacja obiektu wartościami zadanymi może być zrealizowana:

- za pomocą tzw. listy inicjalizacyjnej,
- z zastosowaniem inicjalizacji konstruktorowej.

Lista inicjalizacyjna

Ogólna postać **listy inicjalizacyjnej** (ang. *initializer list*, *initialization list*) jest następująca:

`{wyrażenie_1, wyrażenie_2, ...}`

gdzie `wyrażenie_1`, `wyrażenie_2` itd. stanowią wyrażenia określające zadane wartości początkowe zmiennych członkowskich obiektu.

UWAGA

Wykorzystanie listy inicjalizacyjnej jest możliwe począwszy od standardu C++11.

Ogólna postać wyrażenia pozwalającego na inicjalizację zmiennych członkowskich obiektu przy użyciu listy inicjalizacyjnej jest następująca:

`nazwa_klasy nazwa_obiektu = {wyrażenie_1, wyrażenie_2, ...};`

lub

`nazwa_klasy nazwa_obiektu {wyrażenie_1, wyrażenie_2, ...};`

gdzie:

- `nazwa_klasy` jest identyfikatorem klasy,
- `nazwa_obiektu` jest identyfikatorem obiektu (zmiennej obiektowej),
- `{wyrażenie_1, wyrażenie_2, ...}` to lista inicjalizacyjna.

Pierwsza z zaprezentowanych powyżej postaci stanowi **inicjalizację kopiującą** (ang. *copy initialization*), a druga — **inicjalizację bezpośrednią** (ang. *direct initialization*).

UWAGA

Sposoby inicjalizacji zmiennych należących do typów podstawowych zostały omówione szczegółowo w podrozdziale 3.1 podręcznika.

Wymienione sposoby inicjalizacji zmiennych członkowskich obiektów można opisać również z wykorzystaniem pojęcia listy inicjalizacyjnej. Stąd pierwsza z przedstawionych powyżej postaci jest **nazywana inicjalizacją listową kopiującą** (ang. *copy-list-initialization*), a druga — **inicjalizacją listową bezpośrednią** (ang. *direct-list-initialization*). Inicjalizacja bezpośrednia jest w tym przypadku **inicjalizacją jednolitą** (ang. *uniform initialization*), która jest zalecanym sposobem inicjalizacji obiektów zadanymi wartościami.

UWAGA

Inicjalizacja jednolita jest czasem nazywana **inicjalizacją klamrową** (ang. *brace initialization*).

Inicjalizacja agregacyjna

W najprostszym przypadku, jeśli wszystkie zmienne członkowskie klasy są publiczne i zarazem w klasie nie zdefiniowano żadnego konstruktora, w celu ich inicjalizacji można zastosować tzw. **inicjalizację agregacyjną** (ang. *aggregate initialization*).

Przykład 12.9

```
#include <iostream>
using namespace std;
```

// Definicja klasy Pracownik:

```
class Pracownik {
public:
```

 // Deklaracje zmiennych członkowskich:

```
    int id;
```

```
    string imie;
```

```
    string nazwisko;
```



```

// Prototyp funkcji członkowskiej:
void wyswietlDane();

};

/* UWAGA
 * Klasa Pracownik nie zawiera żadnego konstruktora zdefiniowanego przez programistę.
 */

// Definicja funkcji członkowskiej wyswietlDane():
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
}

int main() {
    // Utworzenie i inicjalizacja agregacyjna obiektu pracownik1:
    Pracownik pracownik1 = {1, "Jan", "Kowalski"};
    /* Uwaga
     * Wszystkie zmienne członkowskie obiektu pracownik1 zainicjowano przy wykorzystaniu inicjalizacji kopiującej
     * z listą inicjalizacyjną.
     */
    pracownik1.wyswietlDane();

    // Utworzenie i inicjalizacja agregacyjna obiektu pracownik2:
    Pracownik pracownik2 {2, "Adam", "Nowak"};
    /* Uwaga
     * Zmienne członkowskie obiektu pracownik2 zainicjowano z zastosowaniem inicjalizacji bezpośredniej
     * z listą inicjalizacyjną.
     */
    pracownik2.wyswietlDane();

    return 0;
}

```

W programie zainicjowano dwa obiekty należące do klasy `Pracownik`. Zastosowano inicjalizację agregacyjną, ponieważ wszystkie zmienne członkowskie klasy `Pracownik` są publiczne oraz nie zawiera ona definicji żadnego konstruktora. Należy zauważyć, że inicjalizacja bezpośrednia z użyciem listy inicjalizacyjnej jest inicjalizacją jednolitą.

Ćwiczenie 12.9

Zmodyfikuj program zawarty w przykładzie 12.9 — w klasie `Pracownik` zdefiniuj dodatkową zmienną członkowską o nazwie `data_zatrudnienia` należącą do typu strukturalnego `Data`. Struktura `Data` powinna zawierać trzy pola umożliwiające zapamiętanie zadanej daty — dnia, miesiąca i roku. Zainicjuj obiekt `pracownik` należący do klasy `Pracownik` wybranymi przez siebie wartościami z wykorzystaniem inicjalizacji agregacyjnej.

Inicjalizacja

Jeśli zmienna
konstruktor
obektu nale
W ogólności
co zostało ju

Przykład 12

```
#include <
using name
```

```
// Definicja kla
class Prac
```

```
// Deklar
int id
string
```

```
public:
```

```
// Proto
```

```
Praco
```

```
// Proto
```

```
Praco
```

```
// Proto
```

```
void
```

```
};
```

```
// Definicje k
```

```
Pracownik
```

```
id =
```

```
imie
```

```
nazw
```

```
}
```

```
Pracowni
```

```
id =
```

```
imie
```

```
nazw
```

```
cout
```

```
}
```

```
// Definicja
```

```
void Pr
```

```
cou
```

```
cou
```

```
cou
```

```
}
```

Inicjalizacja obiektów za pomocą zdefiniowanego konstruktora

Jeśli zmienne członkowskie klasy są prywatne albo programista zdefiniował w klasie własne konstruktory, inicjalizacja agregacyjna nie jest możliwa. W takiej sytuacji w celu inicjalizacji obiektu należy zastosować konstruktor zdefiniowany samodzielnie.

W ogólności wywołanie konstruktora może być zrealizowane w sposób niejawni lub jawny — co zostało już zaprezentowane we wcześniejszych przykładach w tym rozdziale.

Przykład 12.10

```
#include <iostream>
using namespace std;

// Definicja klasy Pracownik:
class Pracownik {
    // Deklaracje prywatnych zmiennych członkowskich:
    int id;
    string imie, nazwisko;

public:
    // Prototyp konstruktora domyślnego:
    Pracownik();
    // Prototyp konstruktora parametrycznego:
    Pracownik(int, string, string);
    // Prototyp funkcji członkowskiej:
    void wyswietlDane();
};

// Definicje konstruktorów:
Pracownik::Pracownik() {
    id = -1;
    imie = "Imię domyślne";
    nazwisko = "Nazwisko domyślne";
}

Pracownik::Pracownik(int pId, string pImie, string pNazwisko) {
    id = pId;
    imie = pImie;
    nazwisko = pNazwisko;

    cout << "Komunikat kontrolny: wywołano konstruktor parametryczny ..."
         << endl;
}

// Definicja funkcji członkowskiej wyswietlDane():
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
}
```



```

int main() {
    // Utworzenie obiektu pracownik1 jako instancji klasy Pracownik:
    Pracownik pracownik1 = {1, "Jan", "Kowalski"}; // inicjalizacja listowa kopiująca
    pracownik1.wyswietlDane();
    cout << endl;

    // Utworzenie obiektu pracownik2 jako instancji klasy Pracownik:
    Pracownik pracownik2 {2, "Adam", "Nowak"}; // inicjalizacja jednolita (klamrowa)
    pracownik2.wyswietlDane();
    cout << endl;

    return 0;
}

```

W programie zostały utworzone dwa obiekty należące do klasy `Pracownik`: `pracownik1` i `pracownik2`. Oba zostały utworzone i zainicjowane przez niejawne wywołanie konstruktora parametrycznego.

W celu inicjalizacji obiektu `pracownik1` zastosowano inicjalizację listową kopiującą: `Pracownik pracownik1 = {1, "Jan", "Kowalski"};`.

Z kolei obiekt `pracownik2` został zainicjowany przy użyciu inicjalizacji listowej bezpośredniej, czyli inicjalizacji jednolitej (klamrowej): `Pracownik pracownik2 {2, "Adam", "Nowak"};`.

Zalecanym sposobem inicjalizacji jest w tym przypadku inicjalizacja jednolita.

Ćwiczenie 12.10

Zmodyfikuj program z przykładu 12.10 — utwórz i zainicjuj obiekty `pracownik1` i `pracownik2` przez jawne wywołanie konstruktora parametrycznego.

Inicjalizacja konstruktorowa

Ogólna postać wyrażenia pozwalającego na inicjalizację zmiennych członkowskich obiektu z wykorzystaniem **inicjalizacji konstruktorowej** (ang. *constructor initialization*) jest następująca:

`nazwa_klasy nazwa_obiektu (wyrażenie_1, wyrażenie_2, ...);`

gdzie:

- `nazwa_klasy` jest identyfikatorem klasy,
- `nazwa_obiektu` jest identyfikatorem obiektu (zmiennej obiektowej),
- `wyrażenie_1, wyrażenie_2, ...` to wyrażenia określające zadane wartości dla kolejnych parametrów konstruktora.

Przykład 12.11

```

#include <iostream>
using namespace std;

// Definicja klasy Pracownik:
class Pracownik {
    // Deklaracje prywatnych zmiennych członkowskich:
    int id;
    string imie, nazwisko;
public:
    // Prototypy konstruktorów:
    Pracownik();
    Pracownik(int, string, string);
    // Prototyp funkcji członkowskiej:
    void wyswietlDane();
};

// Definicje konstruktorów:
Pracownik::Pracownik() {
    id = -1;
    imie = "Imię domyślne";
    nazwisko = "Nazwisko domyślne";
}

Pracownik::Pracownik(int pId, string pImie, string pNazwisko) {
    id = pId;
    imie = pImie;
    nazwisko = pNazwisko;
}

// Definicja funkcji członkowskiej wyswietlDane():
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
}

int main() {
    // Utworzenie obiektu pracownik1 jako instancji klasy Pracownik:
    Pracownik pracownik (1, "Jan", "Kowalski"); // inicjalizacja konstruktorowa
    pracownik.wyswietlDane();

    return 0;
}

```

Klasa Pracownik zawiera m.in. prywatne zmienne członkowskie id, imie i nazwisko oraz definicje konstruktorów, domyślnego i parametrycznego.

Obiekt pracownik został zainicjowany zadanymi wartościami przy użyciu inicjalizacji konstruktorowej.

Ćwiczenie 12.11

Zmodyfikuj program z przykładu 12.11 — utwórz i zainicjuj obiekt `pracownik` przez jawne wywołanie konstruktora parametrycznego. Zmienne członkowskie obiektu `pracownik` zainicjuj z wykorzystaniem inicjalizacji konstruktorowej.

Lista inicjalizacyjna zmiennych członkowskich

W niektórych sytuacjach nie można przypisać zadanych wartości elementom członkowskim klasy w treści konstruktora, np. w przypadku gdy są one zadeklarowane jako stałe `const`. Problem ten można rozwiązać z zastosowaniem listy inicjalizacyjnej zmiennych członkowskich.

Lista inicjalizacyjna zmiennych członkowskich klasy (ang. *class member initializer list*, *class member initialization list*) wchodzi w skład nagłówka konstruktora:

```
nazwa_konstruktor(lista_parametrów) : lista_inicjalizatorów;
```

gdzie:

- `nazwa_konstruktor` jest nazwą (identyfikatorem) klasy,
- `lista_parametrów` to lista parametrów formalnych konstruktora odpowiadających zmiennym członkowskim klasy,
- `lista_inicjalizatorów` to lista inicjalizatorów wybranych/wszystkich zmiennych członkowskich, z których każdy składa się z nazwy zmiennej członkowskiej wraz z przypisaną wartością początkową.

Przykład 12.12

```
#include <iostream>
using namespace std;

// Definicja klasy Pracownik:
class Pracownik {
    const int id = 0; // inicjalizacja stałej const wartością domyślną
    string imie, nazwisko;
public:
    // Prototypy konstruktorów:
    Pracownik();
    Pracownik(int, string, string);
    // Prototyp funkcji członkowskiej:
    void wyswietlDane();
};

// Definicja konstruktora domyślnego z listą inicjalizatorów zmiennych członkowskich:
Pracownik::Pracownik():
    id {-1},
    imie {"Imię domyślne"},
    nazwisko {"Nazwisko domyślne"} {
```

```

cout << "Komunikat kontrolny: wywołano konstruktor domyślny ..." << endl;
}
// Definicja konstruktora parametrycznego z listą inicjalizatorów zmiennych członkowskich:
Pracownik::Pracownik(int pId, string pImie, string pNazwisko):
    id {pId},
    imie {pImie},
    nazwisko {pNazwisko} {
    cout << "Komunikat kontrolny: wywołano konstruktor parametryczny ..."
        << endl;
}
// Definicja funkcji członkowskiej:
void Pracownik::wyswietlDane() {
    cout << "Numer identyfikacyjny: " << id << endl;
    cout << "Imię: " << imie << endl;
    cout << "Nazwisko: " << nazwisko << endl;
}

int main() {
    // Utworzenie obiektu pracownik1 jako instancji klasy Pracownik:
    Pracownik pracownik1;
    /* UWAGA
    * Konstruktor domyślny zdefiniowany przez programistę został wywołany w sposób niejawni.
    * Konstruktor ten można oczywiście wywołać w sposób jawny:
    *   Pracownik pracownik1 = Pracownik();
    * Obiekt pracownik1 został utworzony i zainicjowany wartościami domyślnymi.
    */
    // Wyświetlenie wartości przechowywanych w zmiennych członkowskich obiektu pracownik1:
    pracownik1.wyswietlDane();
    cout << endl;

    // Utworzenie obiektu pracownik2 jako instancji klasy Pracownik:
    Pracownik pracownik2(1, "Jan", "Kowalski");
    /* UWAGA
    * Konstruktor parametryczny zdefiniowany przez programistę został wywołany w sposób niejawni.
    * Obiekt pracownik2 został utworzony i zainicjowany wartościami argumentów podanych w wywołaniu
    *   tego konstruktora.
    * Konstruktor parametryczny można oczywiście wywołać w sposób jawny:
    *   Pracownik pracownik2 = Pracownik(1, "Jan", "Kowalski");
    */
    // Wyświetlenie wartości przechowywanych w zmiennych członkowskich obiektu pracownik2:
    pracownik2.wyswietlDane();

    return 0;
}

```


W programie zdefiniowano klasę `Pracownik`. W klasie tej zadeklarowano i zainicjowano wartość domyślną `0` zmienną członkowską o nazwie `id` jako stałą `const`. Zmienne członkowskie `imie` i `nazwisko` nie zostały zainicjowane przy użyciu inicjatorów wewnątrzklasowych. W klasie `Pracownik` zdefiniowano ponadto dwa konstruktory (domyślny i parametryczny) z listami inicjalizatorów wszystkich zmiennych członkowskich.

Obiekt `pracownik1` został utworzony i zainicjowany wartościami domyślnymi przy wykorzystaniu konstruktora domyślnego z listą inicjalizatorów wszystkich zmiennych członkowskich. Inicjalizacja zmiennych członkowskich (tutaj zmiennej `id`) zrealizowana na poziomie konstruktora ma wyższy priorytet od inicjalizacji bezpośrednio w definicji klasy za pomocą inicjalizatorów wewnątrzklasowych.

Obiekt `pracownik2` utworzono i zainicjowano zadanymi wartościami z zastosowaniem konstruktora parametrycznego, który również zawiera listę inicjalizatorów zmiennych członkowskich.

Ćwiczenie 12.12

Zmodyfikuj program z przykładu 12.12 — zmodyfikuj definicje konstruktorów domyślnego i parametrycznego w taki sposób, aby nadać, odpowiednio, wartość domyślną i wartość zadaną zmiennej członkowskiej `id` typu `const` w ciałach tych konstruktorów, a nie na liście inicjalizatorów zmiennych członkowskich. Czy kompilacja programu zakończy się sukcesem? Uzasadnij, dlaczego rezultaty kompilacji są takie, a nie inne.

12.3. Konstruktor kopiujący

Konstruktor kopiujący (ang. *copy constructor*) to konstruktor przeciążony (ang. *overloaded constructor*), którego zadaniem jest utworzenie i inicjalizacja nowego obiektu należącego do określonej klasy na podstawie innego obiektu będącego instancją tej klasy — obiektu wzorcowego. Nowy obiekt jest identyczny z obiektem wzorcowym (oryginałem) i jednocześnie jest od niego całkowicie niezależny.

Ogólna postać prototypu (deklaracji) konstruktora kopiującego jest następująca:

```
nazwa_klasy (const nazwa_klasy &wzorzec);
```

gdzie `nazwa_klasy` jest identyfikatorem klasy, której instancją jest nowy obiekt, a `wzorzec` to obiekt wzorcowy, na podstawie którego zostanie utworzony nowy obiekt.

UWAGA

Jeśli programista nie zdefiniuje własnego konstruktora kopiującego, kompilator automatycznie utworzy domyślny konstruktor kopiujący.

Konstruktor kopiujący może być wywołany w sposób jawny lub niejawny. Pierwszy przypadek dotyczy sytuacji, w której nowy obiekt jest tworzony na podstawie wywołania konstruktora z argumentem referencyjnym do obiektu wzorcowego traktowanego jako stała `const`. Niejawne wywołanie konstruktora kopiującego zaś jest stosowane, jeśli:

- nowy obiekt jest tworzony na podstawie innego obiektu przy użyciu operatora `=`,
- istniejący obiekt jest parametrem wejściowym dowolnej funkcji przekazywanym do niej przez wartość,
- istniejący obiekt jest zwracany do otoczenia (np. do programu głównego) przez inną (dowolną) funkcję.

Przykład 12.13

```
#include <iostream>
using namespace std;

// Definicja klasy Prostokat:
class Prostokat {
private:
    double bok1;
    double bok2;
public:
    // Prototypy konstruktorów:
    Prostokat(); // konstruktor domyślny
    Prostokat(double, double); // konstruktor z parametrami
    // Prototyp konstruktora kopiującego:
    Prostokat(const Prostokat &); // konstruktor kopiujący
    // Prototypy metod publicznych:
    void pobierzBoki(double&, double&);
    void ustawBoki(double, double);
    double pole();
    double obwod();
};

// Definicje „zwykłych” konstruktorów:
Prostokat::Prostokat() {
    cout << "Nastąpiło wywołanie konstruktora domyślnego ..." << endl;
}
Prostokat::Prostokat(double pBok1, double pBok2) {
    cout << "Nastąpiło wywołanie konstruktora parametrycznego ..." << endl;
    bok1 = pBok1;
    bok2 = pBok2;
}

// Definicja konstruktora kopiującego:
Prostokat::Prostokat(const Prostokat& wzorzec) {
```



```

    cout << "Nastąpiło wywołanie konstruktora kopiującego ..." << endl;
    bok1 = wzorzec.bok1;
    bok2 = wzorzec.bok2;
}
// Definicje metod publicznych należących do klasy Prostokat:
void Prostokat::pobierzBoki(double &pBok1, double &pBok2) {
    pBok1 = bok1;
    pBok2 = bok2;
}
void Prostokat::ustawBoki(double pBok1, double pBok2) {
    bok1 = pBok1;
    bok2 = pBok2;
}
double Prostokat::pole() {
    return bok1 * bok2;
}
double Prostokat::obwod() {
    return 2 * bok1 + 2 * bok2;
}
// Definicja funkcji zewnętrznej, niezależnej od klasy Prostokat:
Prostokat kopiujProstokat(Prostokat prostokat) {
    return prostokat;
}
/* UWAGA
* Parametrem formalnym funkcji kopiujProstokat() jest obiekt klasy Prostokat, który jest przekazywany do funkcji
* przez wartość. Dlatego też na stosie podczas wywołania tej funkcji zostanie wykonana kopia obiektu jako argumentu
* tego wywołania.
* Funkcja zwraca na zewnątrz obiekt typu Prostokat. Tym samym na stosie w trakcie wywołania tej funkcji ponownie
* zostanie wykonana kopia obiektu.
*/

int main() {
    double b1, b2; // zmienne pomocnicze

    cout << "PIERWSZY PROSTOKĄT" << endl;
    // Utworzenie obiektu wzorcowego:
    Prostokat p1(1, 2); // wywołanie konstruktora parametrycznego
    // Wyświetlenie wartości zmiennych członkowskich obiektu p1:
    p1.pobierzBoki(b1, b2);
    cout << "bok1 = " << b1 << endl;
    cout << "bok2 = " << b2 << endl;
    cout << endl;

    cout << "DRUGI PROSTOKĄT" << endl;

```

```

// Utworzenie obiektu p2 na podstawie obiektu p1:
Prostokat p2 = p1;

/* UWAGA
 * Na stosie utworzony zostaje nowy obiekt p2 jako kopia obiektu wzorcowego p1.
 * Konstruktor kopiujący jest wywoływany w sposób niejawni.
 */

p2.pobierzBoki(b1, b2);
cout << "bok1 = " << b1 << endl;
cout << "bok2 = " << b2 << endl;
cout << endl;

cout << "TRZECI PROSTOKAT" << endl;
// Utworzenie obiektu p3 na podstawie obiektu p1:
Prostokat p3(p1);

/* UWAGA
 * Na stosie utworzony zostaje nowy obiekt p3, należący do klasy Prostokat, jako kopia obiektu wzorcowego p1.
 * Konstruktor kopiujący jest wywoływany w sposób jawny — bezpośredni.
 * Argumentem wywołania konstruktora kopiującego jest referencja do obiektu p1, traktowanego jako stała const.
 */

p3.pobierzBoki(b1, b2);
cout << "bok1 = " << b1 << endl;
cout << "bok2 = " << b2 << endl;
cout << endl;

cout << "CZWARTY PROSTOKAT" << endl;
// Utworzenie obiektu p4 przez użycie konstruktora domyślnego:
Prostokat p4;

// Skopiowanie obiektu p1 do obiektu p4:
p4 = kopiujProstokat(p1);

/* UWAGA
 * Konstruktor kopiujący został w tym przypadku wywołany dwukrotnie: pierwszy raz — podczas kopiowania
 * argumentu do funkcji kopiujProstokat(), a drugi — podczas zwracania wartości przez tę funkcję.
 * Obydwa wywołania konstruktora kopiującego są niejawne.
 */

p4.pobierzBoki(b1, b2);
cout << "bok1 = " << b1 << endl;
cout << "bok2 = " << b2 << endl;
cout << endl;

return 0;
}

```

W przykładzie zdefiniowano konstruktor kopiujący należący do klasy Prostokat. Konstruktor ten podczas wykonywania programu jest wywoływany kilkakrotnie — zarówno w sposób jawny, jak i niejawni.

Jawne wywołanie konstruktora kopiującego `Prostokat p3(p1)`; powoduje utworzenie nowego obiektu `p3` jako instancji klasy `Prostokat` — na podstawie obiektu wzorcowego `p1`. Zmienne członkowskie obiektu `p3` zostają zainicjowane wartościami skopiowanymi z elementów członkowskich obiektu `p1`.

Natomiast niejawne wywołanie konstruktora kopiującego następuje przy tworzeniu i inicjalizacji obiektu `p2`: `Prostokat p2 = p1`; Operator `=` oznacza w tym przypadku wywołanie konstruktora kopiującego, a nie przypisanie.

Oprócz tego dwukrotne niejawne wywołanie konstruktora kopiującego następuje przy wywołaniu niezależnej funkcji `kopiujeProstokat()`: `p4 = kopiujeProstokat(p1)`; Obiekt `p1` jest przekazywany do tej funkcji `kopiujeProstokat()` przez wartość, a rezultat wywołania jest zwracany na zewnątrz omawianej funkcji — również przez wartość. Tym samym kopia obiektu `p1` jest wykonywana w pamięci operacyjnej na stosie dwukrotnie.

Ćwiczenie 12.13

Zmodyfikuj program zawarty w przykładzie 12.13 — utwórz i zainicjuj obiekty `p2` oraz `p4` przez jawne, bezpośrednie, wywołanie konstruktora kopiującego. Jako wzorec wykorzystaj obiekt `p1`.

12.4. Delegowanie konstruktorów

W procesie definiowania konstruktorów często się zdarza, że każdy z nich realizuje dokładnie te same czynności (operacje), np. inicjuje te same zmienne członkowskie. Skutkuje to koniecznością powielenia tego samego kodu w każdym konstruktorze — co jest zjawiskiem niepożądanym.

Powielanie tego samego kodu źródłowego w wielu konstruktorach w praktyce można uniknąć na dwa sposoby. Pierwszy z nich polega na zdefiniowaniu dodatkowej funkcji, która zawiera kod źródłowy wspólny dla wielu konstruktorów. Następnie w każdym z konstruktorów zdefiniowaną funkcję należy po prostu wywołać. Drugi sposób opiera się na wykorzystaniu mechanizmu delegowania konstruktora.

Delegowanie konstruktora (ang. *constructor delegation*, *delegating constructor*) polega na wywołaniu jednego konstruktora — tego, który zawiera wspólny kod — przez inny konstruktor należący do tej samej klasy. Delegowany konstruktor może być wywoływany przez wiele innych konstruktorów. W celu wywołania konstruktora zawierającego wspólny kod należy to wywołanie umieścić na liście inicjalizacyjnej innego konstruktora.

Składnia delegacji konstruktora do innego konstruktora jest zgodna z zasadami inicjalizacji zmiennych członkowskich klasy z zastosowaniem listy inicjalizacyjnej.

UWAGA
Delegowa

Poniżej zilu
powielania
stano sposo
wspólny dla
delegowani

Przykład 1

```
#include  
using nam
```

```
// Definicja k  
class Pro  
private:  
    // Dekl  
    doub  
    doub  
    stri  
public:
```

```
// Pro  
Pro  
// Pro  
Pro  
// Pro  
voi  
voi  
};
```

```
// Definicja  
Prostok  
// W  
cou
```

```
// V  
us
```

```
}  
// Definic  
Prosto  
//  
co
```

UWAGA

Delegowanie konstruktorów jest możliwe począwszy od standardu C++11.

Poniżej zilustrowano za pomocą przykładów oba wymienione wcześniej sposoby uniknięcia powielania tego samego kodu w konstruktorach. Przy czym w przykładzie 12.14 wykorzystano sposób polegający na zdefiniowaniu dodatkowej funkcji, która zawiera kod źródłowy wspólny dla wielu konstruktorów. W przykładzie 12.15 z kolei zaimplementowano mechanizm delegowania konstruktora.

Przykład 12.14

```
#include <iostream>
using namespace std;

// Definicja klasy Prostokat:
class Prostokat {
private:
    // Deklaracja prywatnych zmiennych członkowskich:
    double bok1;
    double bok2;
    string kolor;
public:
    // Prototyp konstruktora domyślnego:
    Prostokat();
    // Prototyp konstruktora z parametrami:
    Prostokat(double, double);
    // Prototypy metod publicznych:
    void ustawKolor(string);
    void wyswietlDane();
};

// Definicja konstruktora domyślnego:
Prostokat::Prostokat() {
    // Wyświetlenie komunikatu kontrolnego:
    cout << "Komunikat kontrolny: został wywołany konstruktor domyślny ..."
    << endl;
    // Wywołanie metody ustawKolor():
    ustawKolor("biały"); // Ustalenie wartości domyślnej zmiennej członkowskiej kolor.
}

// Definicja konstruktora parametrycznego:
Prostokat::Prostokat(double pBok1, double pBok2) {
    // Wyświetlenie komunikatu kontrolnego:
    cout << "Komunikat kontrolny: został wywołany konstruktor parametryczny ..."
    << endl;
```



```

// Inicjalizacja zmiennych członkowskich bok1 i bok2:
bok1 = pBok1;
bok2 = pBok2;
// Wywołanie metody ustawKolor():
ustawKolor("biały"); // Ustalenie wartości domyślnej zmiennej członkowskiej kolor.
}
/* UWAGA
 * W celu uniknięcia powielania kodu w zdefiniowanych konstruktorach w każdym z nich wywoływana jest
 * ta sama funkcja: ustawKolor(), z argumentem „biały”.
 * Wartość początkowa „biały” jest traktowana jako wartość domyślna zmiennej członkowskiej kolor.
 */
// Definicje metod publicznych należących do klasy Prostokat:
void Prostokat::ustawKolor(string pKolor) {
    kolor = pKolor;
}
void Prostokat::wyswietlDane() {
    cout << "Pierwszy bok: " << bok1 << endl;
    cout << "Drugi bok: " << bok2 << endl;
    cout << "Kolor: " << kolor << endl;
}

int main() {
    // Utworzenie obiektu p1:
    Prostokat p1;
    // Wyświetlenie parametrów prostokąta p1:
    p1.wyswietlDane();
    cout << endl;

    // Utworzenie obiektu p2:
    Prostokat p2(1, 2);
    // Wyświetlenie parametrów prostokąta p2:
    p2.wyswietlDane();

    // Ustawienie (modyfikacja) wartości zmiennej członkowskiej kolor obiektu p2:
    p2.ustawKolor("czarny");
    /* UWAGA
     * Modyfikacja wartości prywatnej zmiennej członkowskiej kolor została wykonana przez wywołanie publicznej
     * metody dostępowej ustawKolor().
     */
    // Ponowne wyświetlenie parametrów prostokąta p2:
    p2.wyswietlDane();

    return 0;
}

```

W klasie `Prostokat` zdefiniowano metodę publiczną `ustawKolor()`, która pozwala na ustawienie/zmianę wartości zmiennej członkowskiej `kolor`. Wywołanie tej metody w każdym ze zdefiniowanych konstruktorów stanowi ich wspólny kod.

Wywołanie konstruktora domyślnego (`Prostokat p1;`) powoduje zainicjowanie zmiennej członkowskiej `kolor` prostokąta `p1`. Pozostałe zmienne członkowskie, tj. `bok1` i `bok2`, nie zostają zainicjowane. Wywołanie konstruktora parametrycznego (`Prostokat p2(1, 2);`) skutkuje natomiast inicjalizacją wszystkich zmiennych członkowskich obiektu `p2`.

Po wywołaniu metody `ustawKolor()` z argumentem "czarny" następuje zmiana wartości zmiennej członkowskiej `kolor` prostokąta `p2` z wartości "biały" na "czarny".

Ćwiczenie 12.14

Zmodyfikuj program z przykładu 12.14 — zdefiniuj dodatkowo prywatną metodę bezparametrową o nazwie `ustawKolorPocatkowy()`, która będzie zawierać w swoim ciele jedną instrukcję: `kolor = "biały";`. Wywołaj tę metodę w obu konstruktorach zamiast metody publicznej `ustawKolor()`. Zinterpretuj uzyskane rezultaty.

Przykład 12.15

```
#include <iostream>
using namespace std;

// Definicja klasy Prostokat:
class Prostokat {
private:
    // Deklaracja prywatnych zmiennych członkowskich:
    double bok1;
    double bok2;
    string kolor;
public:
    // Prototyp konstruktora domyślnego:
    Prostokat();
    // Prototyp konstruktora z parametrami:
    Prostokat(double, double);
    // Prototypy metod publicznych:
    void ustawKolor(string);
    void wyswietlDane();
};

// Definicja konstruktora domyślnego:
Prostokat::Prostokat() {
    // Wyświetlenie komunikatu kontrolnego:
    cout << "Komunikat kontrolny: został wywołany konstruktor domyślny ..."
    << endl;
    // Inicjalizacja zmiennej członkowskiej kolor:
    kolor = "biały";
}
```


// Definicja konstruktora z parametrami z delegacją konstruktora domyślnego:

```
Prostokat::Prostokat(double pBok1, double pBok2) : Prostokat::Prostokat() {
    // Wyświetlenie komunikatu kontrolnego:
    cout << "Komunikat kontrolny: został wywołany konstruktor parametryczny ..."
        << endl;
    // Inicjalizacja zmiennych członkowskich bok1 i bok2:
    bok1 = pBok1;
    bok2 = pBok2;
}
/* UWAGA
 * W celu uniknięcia powielania kodu, który jest zawarty w konstruktorze domyślnym,
 * wywołanie konstruktora domyślnego umieszczono na liście inicjalizacyjnej konstruktora parametrycznego.
 */
```

// Definicje metod publicznych należących do klasy Prostokat:

```
void Prostokat::ustawKolor(string pKolor) {
    kolor = pKolor;
}
void Prostokat::wyswietlDane() {
    cout << "Pierwszy bok: " << bok1 << endl;
    cout << "Drugie bok: " << bok2 << endl;
    cout << "Kolor: " << kolor << endl;
}
```

```
int main() {
```

 // Utworzenie obiektu p1:

```
    Prostokat p1;
```

 // Wyświetlenie parametrów prostokąta p1:

```
    p1.wyswietlDane();
```

```
    cout << endl;
```

 // Utworzenie obiektu p2:

```
    Prostokat p2(1, 2);
```

 // Wyświetlenie parametrów prostokąta p2:

```
    p2.wyswietlDane();
```

 // Ustawienie (modyfikacja) wartości zmiennej członkowskiej kolor dla prostokąta p2:

```
    p2.ustawKolor("czarny");
```

 /* UWAGA

 * Modyfikacja wartości prywatnej zmiennej członkowskiej kolor została wykonana przez wywołanie

 * publicznej metody dostępowej ustawKolor().

 */

 // Ponowne wyświetlenie parametrów prostokąta p2:

```
    p2.wyswietlDane();
```

```
    return 0;
```

```
}
```

W klasie `Prostokat` zdefiniowano dwa konstruktory: domyślny i parametryczny. Konstruktor domyślny pozwala na inicjalizację wyłącznie jednej zmiennej członkowskiej — `kolor`. Wartość domyślna tej zmiennej została określona jako "biały".

Wywołanie w programie głównym konstruktora domyślnego, `Prostokat p1;`, powoduje zainicjowanie zmiennej członkowskiej `kolor` prostokąta `p1`. Pozostałe zmienne członkowskie, tj. `bok1` i `bok2`, nie zostają zainicjowane.

Wywołanie konstruktora parametrycznego z dwoma argumentami: `1` oraz `2`, skutkuje natomiast inicjalizacją wszystkich zmiennych członkowskich obiektu `p2`: `bok1`, `bok2` i `kolor`. Dzieje się tak dlatego, że przed wywołaniem konstruktora parametrycznego zostaje wywołany „delegowany” konstruktor domyślny, który zawiera inicjalizację zmiennej `kolor`.

Ćwiczenie 12.15

Zmodyfikuj program z przykładu 12.15 — zainicjuj w konstruktorze domyślnym wszystkie zmienne członkowskie klasy `Prostokat`. Program główny pozostaw bez zmian. Zinterpretuj uzyskane rezultaty.

12.5. Destruktory

Destruktor (ang. *destructor*) jest specjalną funkcją członkowską klasy, która jest wykonywana automatycznie podczas niszczenia lub usuwania obiektu. Zadaniem destruktora jest zwolnienie zasobów pamięci operacyjnej przydzielonej obiektowi podczas jego tworzenia i inicjalizacji za pomocą konstruktora.

Ogólna postać definicji destruktora jest następująca:

```
~nazwa_klasy() {  
    zestaw_instrukcji_pomocniczych;  
};
```

gdzie:

- `nazwa_klasy` jest identyfikatorem klasy, która zawiera destruktory,
- `zestaw_instrukcji_pomocniczych` to ciąg instrukcji wykonujących zadania pomocnicze, np. informacyjne.

Definicja destruktora w klasie jest opcjonalna. W przypadku klas, które nie operują na strukturach dynamicznych, plikach, bazach danych, definiowanie destruktora nie jest konieczne. C++ automatycznie zwolni (wyczyści) pamięć zaalokowaną dla wykorzystywanych obiektów.

Destruktory mają pewne szczególne właściwości (cechy), które odróżniają je od zwykłych metod członkowskich klasy, np.:

- w klasie można zdefiniować tylko jeden destruktory,
- destruktory nie ma żadnych parametrów (argumentów),

- destruktor nie zwraca do swojego otoczenia żadnej wartości,
- destruktor nie może być zadeklarowany jako static albo const.

Destruktry są wywoływane automatycznie w sposób niejawny w sytuacji, gdy obiekt jest niszczone lub usuwany, tj.:

- w chwili zakończenia programu (z wyjątkiem awaryjnego zabicia aplikacji),
- w chwili zakończenia wykonywania funkcji, w której obiekt został zdefiniowany (w przypadku inicjalizacji na stosie),
- na końcu bloku kodu, w którym obiekt został zdefiniowany (w przypadku inicjalizacji na stosie),
- w chwili wywołania operatora delete.

Przykład 12.16

```
#include <iostream>
using namespace std;

// Definicja klasy Pracownik:
class Pracownik {
public:
    string imie;
    string nazwisko;
    // Definicje konstruktorów:
    Pracownik() {
        cout << "Komunikat kontrolny: wywołanie konstruktora domyślnego ..."
              << endl;
        imie = "Imię domyślne";
        nazwisko = "Nazwisko domyślne";
    };
    Pracownik(string pImie, string pNazwisko) {
        cout << "Komunikat kontrolny: wywołanie konstruktora parametrycznego ..."
              << endl;
        imie = pImie;
        nazwisko = pNazwisko;
    };
    // Definicja destruktora:
    ~Pracownik() {
        cout << "Komunikat kontrolny: wywołanie destruktoru obiektu ..."
              << endl;
    };
    // Definicja funkcji członkowskiej:
    void wyswietlDane() {
        cout << "Imię: " << imie << endl;
        cout << "Nazwisko: " << nazwisko << endl;
    }
};
```

```

// Definicja zewnętrznej funkcji pomocniczej:
void funkcjaPomocnicza() {
    cout << "Komunikat kontrolny: wywołanie funkcji pomocniczej ..." << endl;
    pracownik pracownik;
    pracownik.wyswietlDane();
}

int main() {
    { // początek bloku kodu
        // Utworzenie obiektu pracownik1 należącego do klasy Pracownik:
        pracownik pracownik1;
        pracownik1.wyswietlDane();
    } // koniec bloku
    /* UWAGA
     * Koniec bloku oznacza koniec życia obiektu pracownik1,
     * co wiąże się z wywołaniem destruktora i zniszczeniem obiektu.
     */
    cout << endl;

    // Utworzenie obiektu typu Pracownik wskazywanego przez wskaźnik w_pracownik2:
    pracownik *w_pracownik2 = new Pracownik("Jan", "Kowalski");
    w_pracownik2->wyswietlDane();
    // Usunięcie obiektu wskazywanego przez wskaźnik w_pracownik2:
    delete w_pracownik2;
    /* UWAGA
     * Użycie operatora delete spowoduje wywołanie destruktora obiektu.
     */
    cout << endl;

    // Wywołanie zewnętrznej funkcji pomocniczej:
    funkcjaPomocnicza();
    cout << endl;
    cout << "Wykonanie pozostałych instrukcji w programie ...";

    return 0;
}

```

Klasa Pracownik zawiera — oprócz innych elementów członkowskich — definicję dwóch konstruktorów i destruktora. Destruktor jest wywoływany w programie trzykrotnie. Każde z tych wywołań jest niejawne.

Pierwsze z wywołań destruktora następuje w chwili wyjścia obiektu pracownik1 poza zakres bloku kodu, w którym obiekt ten został zdefiniowany. Powoduje to zniszczenie obiektu i zwolnienie zasobów w pamięci operacyjnej przydzielonych w celu jego przechowania.

Drugie wywołanie destruktora następuje po użyciu operatora delete, powodującego usunięcie obiektu, dla którego pamięć operacyjna została zaalokowana dynamicznie.

Trzecie, ostatnie wywołanie destruktora następuje w chwili zakończenia działania funkcji zewnętrznej funkcjaPomocnicza(), w której został zdefiniowany obiekt pracownik.

Ćwiczenie 12.6

Zmodyfikuj program z przykładu 12.16 — wywołaj konstruktory obiektów w sposób jawny. Sprawdź, czy ma to wpływ na wywołania destruktorów. Uzasadnij uzyskane rezultaty.

12.6. Pytania i zadania kontrolne

12.6.1. Pytania

1. Co to jest konstruktor? Jakie zadania realizuje?
2. Co to jest konstruktor domyślny? Czym się różni konstruktor domyślny od konstruktora z parametrami?
3. Jaka jest rola konstruktora kopiującego?
4. Na czym polega delegowanie konstruktorów? W jakim celu deleguje się konstruktory?
5. Omów proces inicjowania obiektów z zastosowaniem listy inicjalizacyjnej.
6. Co to jest destruktor? Jakie zadania realizuje?

12.6.2. Zadania

1. Napisz program pozwalający zapamiętać w zmiennych członkowskich obiektu `uczen` dane ucznia: imię, nazwisko, klasę, grupę, numer w dzienniku. Wykorzystaj klasę `Uczen` zawierającą definicje niezbędnych zmiennych członkowskich, konstruktorów — domyślnego i parametrycznego — oraz metody `wyswietlDane()`. Konstruktor domyślny powinien mieć możliwość inicjalizacji wszystkich zmiennych członkowskich wartościami domyślnymi, a konstruktor parametryczny — wartościami zadanymi. Metoda `wyswietlDane()` ma za zadanie wyświetlenie na ekranie monitora wartości przechowywanych w zmiennych członkowskich. Dane wejściowe mają być wprowadzane z klawiatury. Dane po wprowadzeniu i zapisaniu w obiekcie `uczen` powinny być wyświetlane kontrolnie na ekranie monitora. Załóż, że wszystkie elementy członkowskie klasy `Uczen` są publiczne.
2. Napisz program umożliwiający obliczenie objętości, pola powierzchni bocznej oraz długości wszystkich krawędzi prostopadłościanu. Parametry prostopadłościanu mają być wprowadzane z klawiatury, a wyniki wyświetlane na ekranie monitora. Wykorzystaj klasę `Prostopadloscian` zawierającą definicje niezbędnych zmiennych członkowskich, konstruktorów — domyślnego i parametrycznego — oraz funkcji członkowskich. Konstruktor domyślny powinien mieć możliwość inicjalizacji wszystkich zmiennych członkowskich wartościami domyślnymi, a konstruktor parametryczny — wartościami zadanymi.
3. Napisz program pozwalający obliczyć pola powierzchni i obwody pomieszczeń w mieszkaniu. Załóż, że każde z pomieszczeń ma kształt prostokąta, a ich liczba wynosi 4. Wykorzystaj zdefiniowaną samodzielnie klasę `Prostokat`. Obiekty odpowiadające po-

- szczególnym pomieszczeniom w mieszkaniu utwórz i zainicjuj wartościami domyślnymi z użyciem konstruktora kopiującego.
4. Napisz program pozwalający przeliczyć długość wyrażoną w wybranej jednostce angielskiej na długość mierzoną w metrach. Zastosuj zdefiniowaną samodzielnie klasę konstruktora domyślnego i parametrycznego. Zadaniem konstruktora domyślnego jest m.in. wyświetlenie komunikatu informującego użytkownika, że docelową jednostką miary jest metr. Definicja konstruktora parametrycznego powinna zawierać delegację (wywołanie) konstruktora domyślnego. Wykorzystaj obiekt utworzony i zainicjowany przy użyciu konstruktora parametrycznego.
 5. Napisz program pozwalający zapamiętać w obiekcie `samochod` dane samochodu: markę, model, rok produkcji, cenę, numer rejestracyjny, datę pierwszej rejestracji. Wykorzystaj zdefiniowaną samodzielnie klasę `Samochod` zawierającą definicje niezbędnych elementów członkowskich, w tym definicje konstruktora domyślnego i parametrycznego oraz destruktor. Zmienna członkowska reprezentująca datę pierwszej rejestracji samochodu powinna należeć do typu strukturalnego. Zainicjuj obiekt `samochod` będący instancją klasy `Samochod` przy użyciu konstruktora parametrycznego z listą inicjalizacyjną. Dane wejściowe mają być wprowadzane z klawiatury. Dane po wprowadzeniu i zapisaniu w obiekcie `samochod` powinny być wyświetlane kontrolnie na ekranie monitora. Załóż, że wszystkie zmienne członkowskie klasy `Samochod` są prywatne.
 6. Napisz program pozwalający obliczyć odległość pomiędzy dwoma punktami na płaszczyźnie. Dane wejściowe (tj. współrzędne kartezjańskie dwóch punktów) mają być wprowadzane z klawiatury, a wynik wyświetlany na ekranie monitora. Wykorzystaj obiekt `odleglosc` będący instancją klasy `odleglosc`, która zawiera definicje: zmiennych członkowskich typu strukturalnego odpowiadających współrzędnym punktów, konstruktorów — domyślnego i parametrycznego — destruktor oraz metody umożliwiającej obliczenie odległości pomiędzy zadanymi punktami. Zainicjuj obiekt `odleglosc` zadanymi wartościami za pomocą konstruktora parametrycznego z użyciem listy inicjalizacyjnej (wariant pierwszy) i metody konstruktorowej (wariant drugi). Załóż, że wszystkie zmienne członkowskie klasy `odleglosc` są prywatne.