

Andrzej Stasiewicz

ĆWICZENIA



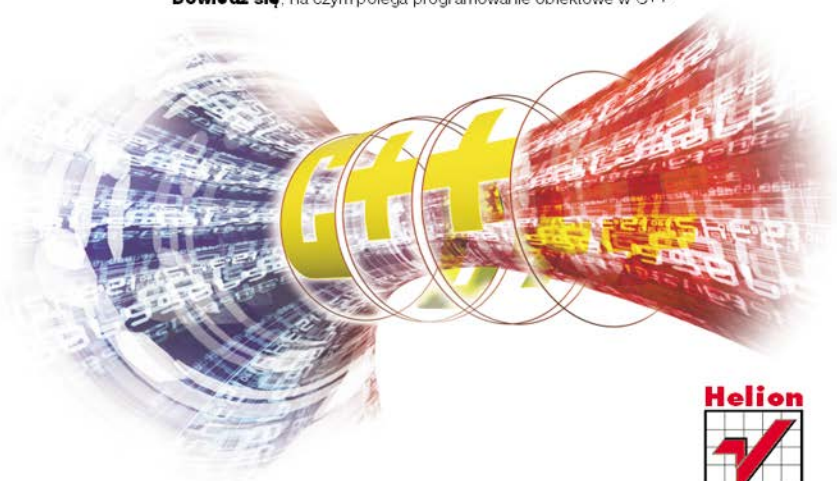
PRAKTYCZNE

# C++

Wydanie III

Przekonaj się, że programowanie w C++ to nic trudnego!

**Poznaj** nowoczesny warsztat pracy programisty  
**Nauč się** tworzyć funkcje i używać typów danych  
**Dowiedz się**, na czym polega programowanie obiektowe w C++



## » Idź do

- Spis treści
- Przykładowy rozdział

## » Katalog książek

- Katalog online
- Zamów drukowany katalog

## » Twój koszyk

- Dodaj do koszyka

## » Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

## » Czytelnia

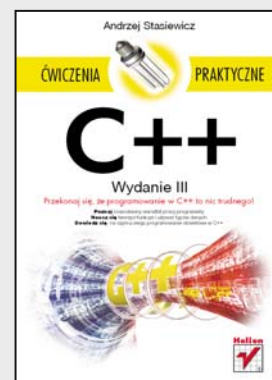
- Fragmenty książek online

## » Kontakt

Helion SA  
ul. Kościuszki 1c  
44-100 Gliwice  
tel. 32 230 98 63  
e-mail: helion@helion.pl  
© Helion 1991–2011

## C++. Ćwiczenia praktyczne. Wydanie III

Autor: [Andrzej Stasiewicz](#)  
ISBN: 978-83-246-3336-4  
Format: 140×208, stron: 160



- Poznaj nowoczesny warsztat pracy programisty
- Naucz się tworzyć funkcje i używać typów danych
- Dowiedz się, na czym polega programowanie obiektowe w C++

### Przekonaj się, że programowanie w C++ to nic trudnego!

C++ stanowi obecnie najbardziej rozpowszechniony język programowania. Choć nie każdy o tym wie, dał on początek wielu innym, wyspecjalizowanym językom, zaś dla tysięcy studentów kierunków informatycznych niezmiennie pozostaje jednym z najważniejszych przedmiotów na studiach. Napisane w nim aplikacje można spotkać dosłownie na każdym kroku i w każdym systemie operacyjnym, a sprawnie posługujący się nim programiści mogą liczyć na dobre oferty pracy. Mimo upływu lat C++ wciąż jest językiem bardzo nowoczesnym, a dzięki długiej historii stanowi narzędzie bardzo dojrzałe i doskonale sprawdzone. Jedynym problemem wydaje się to, że tak wiele osób uważa go za język skomplikowany i trudny do opanowania.

O tym, że wcale tak być nie musi, możesz przekonać się dzięki książce „C++. Ćwiczenia praktyczne. Wydanie III”. Zgromadzone w niej informacje i ćwiczenia w prosty i niezwykle pragmatyczny sposób wprowadzą Cię w podstawy „tajemnej” wiedzy programistycznej. Dowiesz się między innymi, jak posługiwać się plikami źródłowymi tworzącymi program C++, zapewniać sobie dostęp do bibliotek i używać funkcji standardowych, wykonywać operacje wejścia-wyjścia, korzystać z różnych instrukcji warunkowych oraz deklarować czy definiować własne funkcje. Poznasz też standardowe typy danych dostępne w języku C++, nauczysz się deklarować zmienne i przeprowadzać na nich rozmaite operacje. Zrozumiesz, jak tworzyć własne typy danych i do czego może Ci się to przydać.

- Warsztat programisty C++
- Podstawowe informacje o języku i narzędziach
- Organizacja plików źródłowych
- Obsługa strumieni wejścia i wyjścia
- Dołączanie bibliotek i korzystanie z funkcji bibliotecznych
- Implementowanie algorytmów przy użyciu instrukcji warunkowych
- Deklarowanie i definiowanie własnych funkcji
- Tworzenie i używanie zmiennych różnych typów
- Definiowanie własnych typów danych w postaci klas
- Podstawowe informacje o kontenerach

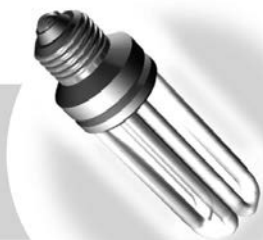
**Przećwicz C++ w praktyce!**

# Spis treści

|   |           |
|---|-----------|
| <b>Wprowadzenie</b>                                 | <b>7</b>  |
| Dlaczego język C++ jest tak ważny?                  | 7         |
| Co da Czytelnikowi ta książka?                      | 8         |
| Co będzie potrzebne do korzystania z książki?       | 9         |
| Jak uczyć się języka z tej książki?                 | 10        |
| <b>Rozdział 1. Nasz programistyczny warsztat</b>    | <b>11</b> |
| <b>Rozdział 2. Nasz pierwszy program</b>            | <b>15</b> |
| Czy to działa?                                      | 15        |
| Sposób na znikanie okienka konsoli                  | 18        |
| Podsumowanie  | 22        |
| <b>Rozdział 3. Pliki źródłowe w języku C++</b>      | <b>23</b> |
| Pliki jako nośniki programów                        | 23        |
| Nośniki programów w C++                             | 24        |
| Dyrektywa #include i scalanie plików cpp i h        | 25        |
| Podsumowanie  | 29        |
| <b>Rozdział 4. Więcej o strumieniach cin i cout</b> | <b>31</b> |
| Standardowe strumienie wejścia i wyjścia            | 31        |
| Kaskadowe posługiwanie się strumieniami             | 34        |
| Odrobina formatowania                               | 36        |
| Odrobina koloru w konsoli                           | 41        |
| Dźwięk w konsoli                                    | 42        |
| Podsumowanie  | 42        |

---

|  |            |
|--|------------|
| <b>Rozdział 5. Przestrzeń na Twoje algorytmy</b>     | <b>45</b>  |
| Początek — najlepsze miejsca na dyrektywy #include   | 45         |
| Po nagłówkach — dostęp do biblioteki standardowej    | 47         |
| Po bibliotece standardowej — nasze własne deklaracje | 48         |
| Funkcja main() — centrum programu                    | 50         |
| Po funkcji main() — definicje innych funkcji         | 53         |
| Podsumowanie   | 54         |
| <b>Rozdział 6. Algorytmy</b>                         | <b>57</b>  |
| Zwrotnica if() ... else ...                          | 57         |
| Zwrotnica switch{...}                                | 63         |
| Pętla for( ...; ...; ...)                            | 69         |
| Pętla while(...)                                     | 75         |
| Pętla do {...} while(...)                            | 78         |
| Instrukcje break i continue                          | 80         |
| Podsumowanie   | 85         |
| <b>Rozdział 7. Funkcje</b>                           | <b>87</b>  |
| Deklarowanie funkcji                                 | 87         |
| Definiowanie funkcji                                 | 88         |
| Argumenty funkcji i referencja                       | 96         |
| Trochę zabawy z dźwiękiem                            | 101        |
| Podsumowanie   | 102        |
| <b>Rozdział 8. Dane</b>                              | <b>103</b> |
| Typy danych  | 103        |
| Deklarowanie oraz inicjowanie prostych danych        | 106        |
| Deklarowanie oraz inicjowanie danych tablicowych     | 108        |
| Deklarowanie oraz inicjowanie danych wskaźnikowych   | 113        |
| Operacje na danych                                   | 119        |
| Podsumowanie   | 126        |
| <b>Rozdział 9. Klasy i obiekty</b>                   | <b>127</b> |
| Klasa jako nowy typ danych                           | 127        |
| Wewnętrzny ustrój klasy — dane                       | 129        |
| Wewnętrzny ustrój klasy — algorytmy                  | 133        |
| Pewien specjalny algorytm, zwany konstruktorem       | 137        |
| Podsumowanie   | 145        |
| <b>Rozdział 10. Kontenery na dane</b>                | <b>147</b> |
| Podsumowanie   | 157        |
| <b>Zakończenie</b>                                   | <b>159</b> |



# 7

## Funkcje



Wydzielone i odpowiednio zatytułowane skrawki algorytmów nazwiemy funkcjami. Przykładem takich wydzielonych algorytmów są powszechnie znane i już wielokrotnie tutaj przywoływane funkcje `sin()`, `cos()` czy `rand()`.

Z jedną funkcją spotykamy się od samego początku — to funkcja `main()`, wbudowana na stałe w język C++. Czy możemy jednak wprowadzać do gry swoje własne funkcje, czy też jesteśmy ograniczeni do użytkowania funkcji bibliotecznych dostarczonych wraz ze środowiskiem? Oczywiście, że możemy deklarować i definiować własne funkcje. Jest to jeden z najważniejszych elementów umiejętności programowania.

## Deklarowanie funkcji

Każda funkcja musi być zadeklarowana (zapowiedziana) przed jej pojawieniem się w programie. W rozdziale 5. (porównaj ćwiczenie 5.8) zwracaliśmy uwagę, że w prostych (jednoplikowych) programach funkcje należy deklarować w przestrzeni między frazą dołączania algorytmów biblioteki standardowej a nagłówkiem funkcji `main()`.

### Ć W I C Z E N I E

#### 7.1 Deklarowanie (zapowiadanie) funkcji

Zadeklaruj rodzinę funkcji `suma()`, obliczającą sumę dwóch, trzech i czterech argumentów typu całkowitego:



Ten program jeszcze nie działa.

```
#include <iostream>
using namespace std;
int suma( int a, int b);
int suma( int a, int b, int c);
int suma( int a, int b, int c, int d);
//-----
int main()
{
    cout << "2 + 3 = " << suma( 2, 3) << endl;
    cout << "2 + 3 + 4 = " << suma( 2, 3, 4) << endl;
    cout << "2 + 3 + 4 + 5 = " << suma( 2, 3, 4, 5);
    char c;
    cin >> c;
}
```

W przykładzie tym wyraźnie widzimy trzy linie deklarujące nowe funkcje. W funkcji `main()` z kolei widzimy wywołania tych funkcji. Wywołania te są zgodne z ich zapowiedziami.

Niestety, ten program nie działa (rysunek 7.1). Kompilacja kończy się komunikatami: Brak ciała funkcji `suma( int , int)`, Brak ciała funkcji `suma( int, int, int)`, Brak ciała funkcji `suma( int , int, int, int)`. Środowisko programistyczne nie znalazło algorytmów, które poprawnie zadeklarowaliśmy i poprawnie wywołaliśmy w funkcji `main()`. Wydaje się to oczywiste, bo przecież nigdzie nie zdefiniowaliśmy tych algorytmów. Nie napisaliśmy ustrojów funkcji.



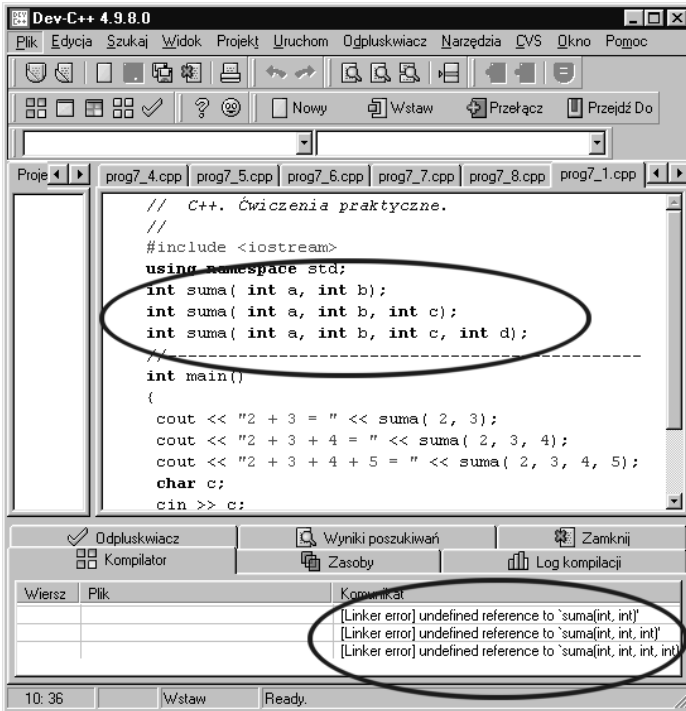
**Częsty błąd:** wprowadzenie do programu deklaracji funkcji, ale zapomnienie o konieczności spisania jej ciała.

## Definiowanie funkcji

### Ć W I C Z E N I E

#### 7.2 Definiowanie (spisywanie) ciał funkcji

Uzupełnij poprzedni przykład, definiując ciała funkcji, które zostały tam zadeklarowane:



**Rysunek 7.1.** Program się nie kompiluje, bo brakuje w nim definicji (czyli tzw. ciała) funkcji. Są tylko ich deklaracje

```
// Tutaj umieść w całości treść poprzedniego przykładu,
// czyli wklejenie nagłówek, udostępnienie biblioteki standardowej,
// deklaracje nowych funkcji i definicję funkcji main()
int suma( int a, int b)
{
    return a + b;
}
//-----
int suma( int a, int b, int c)
{
    return a + b + c;
}
//-----
int suma( int a, int b, int c, int d)
{
    return a + b + c + d;
}
```

Przykład ten tym różni się od poprzedniego (nie działającego), że zawiera definicje (czyli ustroje) wszystkich zadeklarowanych tam i użytkowanych funkcji. Jest to kompletny program z funkcjami użytkownika. Wynik działania programu na rysunku 7.2.

### Rysunek 7.2.

Oto wyniki  
ostatniego  
programu

```

M6 prog7_2
2 + 3 = 5
3 + 4 = 9
3 + 4 + 5 = 14
1

```



**Częsty błąd:** zdefiniowanie ciała funkcji, ale zapomnienie o konieczności jej wcześniejszego zadeklarowania (porównaj ćwiczenie 5.9).

Postawienie średnika za nagłówkiem funkcji, a przed klamrami grupującymi jej ciało.

Niezgodność nagłówka ciała funkcji z jej wcześniejszą deklaracją.

## Ć W I C Z E N I E

### 7.3 Deklarowanie, definiowanie i wywołanie funkcji

Zadeklaruj, zdefiniuj i wywołaj funkcję `parabola()`, która oddaje wartość trójmianu kwadratowego dla konkretnych jego współczynników `a`, `b` i `c`:

```

#include <iostream>
using namespace std;
double parabola( double x, double a, double b, double c);
//-----
int main()
{
    double a, b, c, x, y;
    cout << "Podaj 3 współczynniki paraboli a, b, c: ";
    cin >> a >> b >> c;
    cout << "Podaj x, dla którego należy wyliczyć wartość paraboli: ";
    cin >> x;
    y = parabola( x, a, b, c);
    cout << a << " * x * x + " << b << " * x + " << c << " = " << y;
    char cc;
    cin >> cc;
}
//-----
double parabola( double x, double a, double b, double c)
{
    return a * x * x + b * x + c;
}

```



Funkcja `parabola()` przyjmuje 4 argumenty, wylicza matematyczną wartość i oddaje ją programowi głównemu za pomocą frazy `return`. Wynik na rysunku 7.3.

**Rysunek 7.3.**  
Oto wyniki  
tego programu

```

Mg prog7_3
Podaj 3 współczynniki paraboli a, b, c: 1 2 3
Podaj x, dla którego wyliczyć wartość paraboli: 1
1 * x * x + 2 * x + 3 = 6

```

Nie wszystkie funkcje muszą mieć argumenty — wcześniej spotkaliśmy się z bezargumentową funkcją `rand()` (porównaj ćwiczenie 6.18). Nie wszystkie funkcje muszą też wyliczać i oddawać programowi jakiegokolwiek wartości.



**Częsty błąd:** niezgodność typu danej, oddawanej za pomocą frazy `return`, z typem zapowiedzianym w deklaracji.

## Ć W I C Z E N I E

### 7.4 Deklarowanie, definiowanie i wywoływanie funkcji

Zadeklaruj, zdefiniuj i wywołaj funkcję wyprowadzającą Twoje dane osobowe:

```

#include <iostream>
using namespace std;
void twoje_dane( void);
//-----
int main()
{
    twoje_dane();
    cout << endl << endl;
    twoje_dane();
    char c;
    cin >> c;
}
//-----
void twoje_dane( void)
{
    cout << "Imie i nazwisko: Jan Kowalski" << endl;
    cout << "Wiek           : 23" << endl;
    cout << "Zawod          : Programista AI" << endl;
}

```

Funkcje warto wyodrębniać szczególnie wtedy, gdy będziemy je wielokrotnie wywoływać — tak jak w tym programie dwukrotnie

wywołaliśmy funkcję podającą dane osobowe, zamiast każdorazowo przytaczać odpowiedni fragment algorytmu. Wynik działania programu na rysunku 7.4.

**Rysunek 7.4.**  
*Ten program  
 generuje takie  
 oto informacje*

```
MS
$ prog7_4
Imie i nazwisko: Jan Kowalski
Wiek      : 23
Zawod     : Programista AI

Imie i nazwisko: Jan Kowalski
Wiek      : 23
Zawod     : Programista AI
```

Funkcje wyodrębniamy też wtedy, gdy za pomocą tego samego algorytmu wyliczamy wiele różnych wartości.

## Ć W I C Z E N I E

### 7.5 Funkcja wyliczająca średnią arytmetyczną

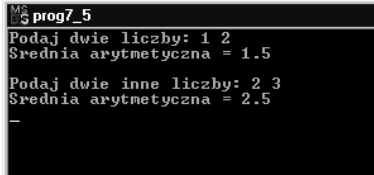
Zadeklaruj, zdefiniuj i wywołaj funkcję wyliczającą średnią arytmetyczną dwóch podanych liczb:

```
#include <iostream>
using namespace std;
double srednia( double a, double b);
//-----
int main()
{
    double q, w;
    cout << "Podaj dwie liczby: ";
    cin >> q >> w;
    cout << "Srednia arytmetyczna = " << srednia( q, w) << endl << endl;
    cout << "Podaj dwie inne liczby: ";
    cin >> q >> w;
    cout << "Srednia arytmetyczna = " << srednia( q, w);
    char c;
    cin >> c;
}
//-----
double srednia( double a, double b)
{
    return (a + b) / 2;
}
```

Wynik działania programu na rysunku 7.5.

**Rysunek 7.5.**

Oto ekran  
wyjściowy



```
MS prog7_5
Podaj dwie liczby: 1 2
Srednia arytmetyczna = 1.5
Podaj dwie inne liczby: 2 3
Srednia arytmetyczna = 2.5
_
```

## Ć W I C Z E N I E

**7.6** Funkcja określająca płeć

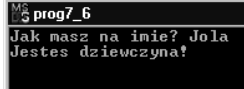
Zadeklaruj, zdefiniuj i wywołaj funkcję określającą płeć osoby o podanym imieniu (porównaj ćwiczenie 6.5):

```
#include <iostream>
#include <string>
using namespace std;
bool czy_dziewczyna( string imie);
//-----
int main()
{
    string txt;
    cout << "Jak masz na imie? ";
    cin >> txt;
    if( czy_dziewczyna( txt))
    {
        cout << "Jestes dziewczyna!";
    }
    else
    {
        cout << "Jestes chlopakiem?";
    }
    char c;
    cin >> c;
}
//-----
bool czy_dziewczyna( string imie)
{
    char ostatni_znak = imie[ imie.size() - 1];
    if( ostatni_znak == 'a')
        return true;    // dziewczyna
    else
        return false;    // chlopak
}
```

Wynik działania programu na rysunku 7.6.

**Rysunek 7.6.**

Oto ekran  
wyjściowy



```
MS
prog7_6
Jak masz na imie? Jola
Jestes dziewczyna!
```

Ta funkcja jako argument otrzymuje od funkcji nadrzędnej tekst, bada ostatni jego znak i na tej podstawie zgaduje, czy ma do czynienia z imieniem żeńskim. Jeśli tak, zwraca wartość logiczną (czyli typu bool) równą true.

Podstawową zaletą wyodrębniania algorytmów w oddzielne funkcje jest możliwość wielokrotnego ich wywoływania z różnymi wartościami argumentów. Ale zdarza się też wprowadzać funkcję do gry tylko dlatego, że podnosi to czytelność programu.

## Ć W I C Z E N I E

**7.7** Funkcje pobierające tekst z klawiatury

Usprawnij pytania o imię i nazwisko, wprowadzając do programu odpowiednie funkcje:

```
string spytaj_o_imie( void);
string spytaj_o_nazwisko( void);
//-----
int main()
{
    string si, sn;
    si = spytaj_o_imie();
    sn = spytaj_o_nazwisko();
    cout << "Nazywasz sie " << si << " " << sn;
    char c;
    cin >> c;
}
//-----
string spytaj_o_imie( void)
{
    string imie;
    cout << "Jak masz na imie? ";
    cin >> imie;
    return imie;
}
//-----
string spytaj_o_nazwisko( void)
{
    string nazwisko;
    cout << "Jakie masz nazwisko? ";
```

```

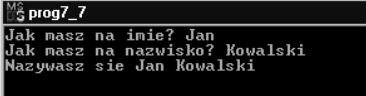
    cin >> nazwisko;
    return nazwisko;
}

```

Wynik działania programu na rysunku 7.7.

### Rysunek 7.7.

Oto ekran  
wyjściowy



```

MS prog7_7
Jak masz na imie? Jan
Jak masz na nazwisko? Kowalski
Nazywasz sie Jan Kowalski

```

Dwie widoczne w powyższym fragmencie kodu funkcje prawdopodobnie wyodrębniono tylko po to, by podnieść czytelność zapisu we wnętrzu funkcji `main()`. Zamiast definiować funkcje, można by po prostu przytoczyć ich ciała bezpośrednio w funkcji `main()`.

## Ć W I C Z E N I E

### 7.8 Funkcja zatrzymująca okienko konsoli

Zadeklaruj, zdefiniuj i wywołaj funkcję o nazwie `stop()`, która poradzi sobie z problemem znikającej konsoli, opisanym w ćwiczeniach 2.2 i 2.3:

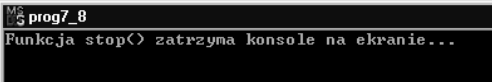
```

void stop( void);
//-----
int main()
{
    cout << "Funkcja stop() zatrzyma konsolę na ekranie...";
    stop();
}
//-----
void stop( void)
{
    char c;
    cin >> c;
}

```

### Rysunek 7.8.

Oto wynik działania  
programu



```

MS prog7_8
Funkcja stop() zatrzyma konsolę na ekranie...

```

Przykład ten (wynik jego działania na rysunku 7.8) dobitnie ilustruje, że warto też wprowadzać do gry funkcje wywoływane tylko jeden raz, ale za to podnoszące czytelność programu. Możemy je nazwać ładnymi opakowaniami brzydkiego kodu.

# Argumenty funkcji i referencja

Teraz dokładniej zbadajmy zagadnienie argumentów funkcji, czyli danych przekazywanych do jej wnętrza.

## ĆWICZENIE

### 7.9 Gdy funkcja modyfikuje swój argument

Napisz prosty przykład z jednoargumentową funkcją, która wypisuje wartość swojego rzeczywistego argumentu na ekranie. Przykład ten za chwilę posłuży do dalszych testów (uwaga — przytaczamy tylko kluczowe fragmenty algorytmów):

```
void zmniejsz_o_3_i_wypisz( double a);
//-----
int main()
{
    double r = 3.14;
    cout << "liczba = " << r << endl;
    zmniejsz_o_3_i_wypisz( r);
    cout << "liczba = " << r;
    char c;
    cin >> c;
}
//-----
void zmniejsz_o_3_i_wypisz( double a)
{
    a = a - 3;
    cout << "liczba zmniejszona o 3 = " << a << endl;
}
```

Dokładnie przeanalizujemy ten i następny przykład, bo za chwilę dowiemy się czegoś bardzo ważnego o argumentach funkcji. W funkcji `main()` najpierw wypisujemy wartość zmiennej `r` i na ekranie (rysunek 7.9) otrzymujemy oczekiwane 3.14. Potem wywołujemy funkcję, która jako argument otrzymuje tę zmienną, ma ją zmniejszyć o 3 i znów wypisać na ekranie, gdzie po chwili otrzymujemy oczekiwane 0.14. Po zakończeniu pracy funkcji ponownie wypisujemy na ekranie zmienną `r` i otrzymujemy nie 0.14, a 3.14.

**Rysunek 7.9.**

Oto wynik  
działania  
programu

```

MS prog7_9
liczba = 3.14
liczba zmniejszona o 3 = 0.14
liczba = 3.14

```

Co w tym zagadkowego? Zagadkowy jest tu *mechanizm ochrony danych*, przekazywanych do wnętrza funkcji. Nasza funkcja przerobiła daną z wartości 3.14 na 0.14, a mimo to stan zmiennej *r* przed wywołaniem funkcji i po jej wywołaniu nie zmienił się.

Funkcje nie pracują na oryginałach, ale *kopiują* sobie argumenty. Wartości argumentów wewnątrz funkcji mogą być modyfikowane, ale to nie wpłynie na ich wartości poza funkcją.

Proces ten może jednak przebiegać zupełnie inaczej...

## Ć W I C Z E N I E

**7.10 Gdy funkcja modyfikuje swój referencyjny argument**

Zmodyfikuj poprzedni przykład tak, aby funkcja otrzymała argument w procesie referencji, a nie kopiowania:

```

void zmniejsz_o_3_i_wypisz( double &a);
//-----
int main()
{
    double r = 3.14;
    cout << "liczba = " << r << endl;
    zmniejsz_o_3_i_wypisz( r);
    cout << "liczba = " << r;
    char c;
    cin >> c;
}
//-----
void zmniejsz_o_3_i_wypisz( double &a)
{
    a = a - 3;
    cout << "liczba zmniejszona o 3 = " << a << endl;
}

```

Trudno dostrzec, na czym polega różnica (wynik działania programu na rysunku 7.10), ale ta funkcja pracowała nie *na kopii* danej *r*, a na jej *oryginale*. Świadczy o tym fakt, że zmniejszenie wartości danej o 3 obowiązuje nie tylko we wnętrzu funkcji, ale w całym programie poniżej tego wywołania.

**Rysunek 7.10.**

Oto wynik  
działania  
programu

```

MS prog7_10
liczba = 3,14
liczba zmniejszona o 3 = 0.14
liczba = 0.14
-

```

Funkcja pracuje na oryginałach swoich argumentów (a nie na ich kopiach), gdy argumenty są przekazywane referencyjnie, czyli gdy przy ich nazwach w deklaracji i nagłówku definicji znajduje się symbol &.

## Ć W I C Z E N I E

**7.11 Funkcja zeruje podane argumenty**

Napisz funkcję, która otrzyma kilka argumentów i je wyzeruje:

```

void wyzeruj( int &a, int &b, int &c);
//-----
int main()
{
    int i, j, k;
    cout << "Przed wywołaniem funkcji: i = " << i << ", j = " << j << ",
    ↵k = " << k << endl;
    wyzeruj( i, j, k);
    cout << "Po wywołaniu funkcji: i = " << i << ",
    ↵j = " << j << ", k = " << k;
    char c;
    cin >> c;
}
//-----
void wyzeruj( int &a, int &b, int &c)
{
    cout << "Zerowanie argumentow wewnatrz funkcji ..." << endl;
    a = b = c = 0;
}

```

```

MS prog7_11
Przed wywołaniem funkcji: i = 4370436, j = 4370432, k = 4198592
Zerowanie argumentow wewnatrz funkcji ...
Po wywołaniu funkcji: i = 0, j = 0, k = 0
-

```

**Rysunek 7.11.** Wynik działania programu

W pierwszej linii wypisujemy wartości zmiennych bezpośrednio po ich zadeklarowaniu. Ponieważ język C++ nie inicjuje zmiennych (więcej na ten temat w następnym rozdziale), w zadeklarowanych zmiennych



znajdują się po prostu śmieci. W następnej linii zmienne te są *przekazywane przez referencję* do wnętrza funkcji. Oznacza to, że funkcja pracuje na oryginałach zmiennych, a nie na ich kopiach, i że wyzerowanie wartości ma szerszy zasięg niż tylko ciało funkcji.

Przykład ten ilustruje, do czego zazwyczaj używa się referencyjnego przekazu argumentów — do modyfikowania większej ich liczby za pomocą jednej funkcji.

Łatwo też się domyślić, że referencja także jest szybsza niż kopiowanie, chociaż zazwyczaj różnica jest na tyle mała, iż dotyczy ekstremalnych zastosowań programowania.

## Ć W I C Z E N I E

**7.12** Funkcja nadużywająca argumentów referencyjnych

Postąpimy teraz źle. Napisz funkcję `kwadrat()`, która dzięki referencji otrzyma argument rzeczywisty i kanałem referencyjnym zwróci wartość drugiej potęgi swojego argumentu:

```
void kwadrat( double &a);
//-----
int main()
{
    double a;
    cout << "Podaj liczbę: ";
    cin >> a;
    kwadrat( a);
    cout << "Kwadrat Twojej liczby wynosi " << a;
    char c;
    cin >> c;
}
//-----
void kwadrat( double &a)
{
    a = a * a;
}
```

Ten program — choć działa dobrze (rysunek 7.12) — prawdopodobnie nadużywa referencji. Nie izoluje wnętrza funkcji, nie zmusza jej do wykonania kopii argumentu, nie wykorzystuje też naturalnej możliwości zwracania wartości przez funkcję. Jednak formalnie nie można mu niczego zarzucić.

**Rysunek 7.12.**

Wynik działania programu

```
MS prog7_12
Podaj liczbę: 4
Kwadrat Twojej liczby wynosi 16
_
```

## Ć W I C Z E N I E

**7.13** Funkcja zwracająca jakąś wartość

Napisz funkcję `kwadrat()`, która otrzyma argument rzeczywisty i w naturalny sposób zwróci wartość drugiej potęgi swojego argumentu:

```
double kwadrat( double a);
//-----
int main()
{
    double a;
    cout << "Podaj liczbę: ";
    cin >> a;
    cout << "Kwadrat Twojej liczby wynosi " << kwadrat( a);
    char c;
    cin >> c;
}
//-----
double kwadrat( double a)
{
    return a * a;
}
```

**Rysunek 7.13.**

Wynik działania programu

```
MS prog7_13
Podaj liczbę: 3
Kwadrat Twojej liczby wynosi 9
_
```

Naturalnym kanałem, za pomocą którego funkcja zwraca pojedynczą wartość, jest zakończenie jej ciała frazą `return`, czyli „oddaj”. Tak skonstruowane funkcje mogą być wbudowywane w różne wyrażenia. Natomiast kanały referencyjne są usprawiedliwione wtedy, gdy danych do przekazania jest dużo (jak np. w ćwiczeniu 7.10) albo gdy przekaz argumentów do ciała funkcji musi być ekstremalnie szybki.

## Przećwicz C++ w praktyce!



C++ stanowi obecnie najbardziej rozpowszechniony język programowania. Choć nie każdy o tym wie, dał on początek wielu innym, wyspecjalizowanym językom, zaś dla tysięcy studentów kierunków informatycznych niezmiennie pozostaje jednym z najważniejszych przedmiotów na studiach. Napisane w nim aplikacje można spotkać dosłownie na każdym kroku i w każdym systemie operacyjnym, a sprawnie posługujący się nim programiści mogą liczyć na dobre oferty pracy. Mimo upływu lat C++ wciąż jest językiem bardzo nowoczesnym, a dzięki długiej historii stanowi narzędzie bardzo dojrzałe i doskonale sprawdzone. Jedynym problemem wydaje się to, że tak wiele osób uważa go za język skomplikowany i trudny do opanowania.

O tym, że wcale tak być nie musi, możesz przekonać się dzięki książce „C++. Ćwiczenia praktyczne. Wydanie III”. Zgromadzone w niej informacje i ćwiczenia w prosty i niezwykłe pragmatyczny sposób wprowadzą Cię w podstawy „tajemnej” wiedzy programistycznej. Dowiesz się między innymi, jak posługiwać się plikami źródłowymi tworzącymi program C++, zapewniać sobie dostęp do bibliotek i używać funkcji standardowych, wykonywać operacje wejścia-wyjścia, korzystać z różnych instrukcji warunkowych oraz deklarować czy definiować własne funkcje. Poznasz też standardowe typy danych dostępne w języku C++, nauczysz się deklarować zmienne i przeprowadzać na nich rozmaite operacje. Zrozumiesz, jak tworzyć własne typy danych i do czego może Ci się to przydać.

- Warsztat programisty C++
- Podstawowe informacje o języku i narzędziach
- Organizacja plików źródłowych
- Obsługa strumieni wejścia i wyjścia
- Dołączanie bibliotek i korzystanie z funkcji bibliotecznych
- Implementowanie algorytmów przy użyciu instrukcji warunkowych
- Deklarowanie i definiowanie własnych funkcji
- Tworzenie i używanie zmiennych różnych typów
- Definiowanie własnych typów danych w postaci klas
- Podstawowe informacje o kontenerach



**Helion**

Nr katalogowy: 6153



Księgarnia internetowa:  
<http://helion.pl>



Zamówienia telefoniczne:  
**0 801 339900**



**0 601 339900**

Sprawdź najnowsze promocje:

• <http://helion.pl/promocje>

Książki najchętniej czytane:

• <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

• <http://helion.pl/nowosci>

**Helion SA**

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

<http://helion.pl>

**helion.pl**  
księgarnia  
internetowa

**Cena 29,00 zł**

ISBN 978-83-246-3336-4

