

# Słowniczek

„Często kilka dobrze dobranych słów  
jest wartych tysiąca obrazów.”

— Anonim

**N**iniejszy *słowniczek* to zbiór terminów używanych w książce. Jest to krótka lista definicji, które są najbardziej potrzebne początkującym programistom. Podobną rolę spełniają także indeks na końcu książki i sekcja *Terminologia* w każdym rozdziale. Bogatszy słowniczek pojęć związanych z językiem C++ znajduje się na stronie [www.stroustrup.com/glossary.html](http://www.stroustrup.com/glossary.html). Poza tym w internecie można znaleźć mnóstwo różnych specjalistycznych glosariuszy różnej jakości. Należy pamiętać, że wiele terminów ma kilka podobnych znaczeń (w niektórych przypadkach wymieniamy po kilka) oraz że większość opisanych przez nas terminów ma także podobne (często w małym stopniu) znaczenia w innych kontekstach. Nie zajmujemy się na przykład definiowaniem pojęcia **abstrakcyjny** w malarstwie, prawie czy filozofii.

**abstrakcja** (ang. *abstraction*) Opis czegoś, w czym celowo i wybiórczo (ukrywa się) ignoruje się szczegóły (np. szczegóły implementacyjne). Jest to rodzaj wybiórczej ignorancji.

**adres** (ang. *address*) Wartość umożliwiająca znalezienie czegoś w pamięci.

**algorytm** (ang. *algorithm*) Procedura rozwiązywania problemu. Skończony zestaw instrukcji, których wykonanie prowadzi do uzyskania wyniku.

**alias** (ang. *alias*) Alternatywny sposób odwoływania się do czegoś. Często jest to nazwa, wskaźnik lub referencja.

**aplikacja** (ang. *application*) Program lub zbiór programów traktowany przez użytkowników jako jednostka.

**argument** (ang. *argument*) Wartość przekazywana funkcji lub szablonowi, w których jest pobierana za pośrednictwem parametrów.

**asercja** (ang. *assertion*) Instrukcja będąca twierdzeniem, że coś w danym miejscu w programie musi być prawdziwe.

**bajt** (ang. *byte*) Podstawowa jednostka adresowa w większości komputerów. Najczęściej jeden bajt zawiera 8 bitów.

**biblioteka** (ang. *library*) Zbiór typów, funkcji, klas itp. implementujących zestaw narzędzi (abstrakcji), które są przeznaczone do użytku w wielu programach.

**bit** (ang. *bit*) Podstawowa komputerowa jednostka informacji. Bit może przyjmować wartości 0 i 1.

**błąd** (ang. *error*) Niezgodność między rozsądnymi oczekiwaniami dotyczącymi działania programu (często wyrażonymi w postaci wymagań lub przewodnika użytkowników) a tym, co program rzeczywiście robi.

**bug, błąd** (ang. *bug*) Błąd w programie.

**czas życia, czas trwania** (ang. *lifetime*) Czas od zainicjowania obiektu do momentu, w którym stał się bezużyteczny (znalazł się poza zakresem dostępności, został usunięty lub zamknięto program).

**dane** (ang. *data*) Wartości wykorzystywane w obliczeniach.

**dane wejściowe** (ang. *input*) Wartości używane w procesie przetwarzania (np. argumenty funkcji lub znaki wpisywane za pomocą klawiatury).

**dane wyjściowe** (ang. *output*) Wartości wytworzone w procesie przetwarzania (np. wynik działania funkcji lub wiersze znaków wydrukowane na ekranie).

**debugowanie** (ang. *debugging*) Czynność polegająca na znajdowaniu i usuwaniu błędów programu. Zazwyczaj debugowanie jest znacznie mniej systematyczne niż testowanie.

**definicja** (ang. *definition*) Deklaracja jednostki zawierająca wszystkie informacje potrzebne do ukończenia programu, który jej używa. Uproszczona definicja: deklaracja alokująca pamięć.

**deklaracja** (ang. *declaration*) Określenie nazwy i jej typu w programie.

**destruktor** (ang. *destructor*) Operacja, która jest niejawnie wywoływana przy usuwaniu obiektu (np. pod koniec zakresu dostępności). Często zwalnia zasoby.

**funkcja** (ang. *function*) Nazwany fragment kodu, który można wywoływać w różnych miejscach programu. Logiczna jednostka przetwarzania.

**funkcja czysto wirtualna** (ang. *pure virtual function*) Funkcja wirtualna, która musi być przesłonięta w klasie pochodnej.

**funkcja wirtualna** (ang. *virtual function*) Funkcja składowa, którą można przesłonić w podklasie.

**hermetyzacja** (ang. *encapsulation*) Chronienie elementów prywatnych (np. szczegółów implementacyjnych) przed nieuprawnionym dostępem.

**ideał** (ang. *ideal*) Perfekcyjna postać, do której się dąży. Zazwyczaj trzeba godzić się na kompromisy i zadowalać się efektem przybliżonym do ideału.

**implementacja** (ang. *implementation*) (1) Pisanie i testowanie kodu. (2) Kod programu.

**inicjować** (ang. *initialize*) Nadać obiektowi pierwszą (początkową) wartość.

**interfejs** (ang. *interface*) Deklaracja lub zbiór deklaracji określających sposób wywoływania fragmentu kodu (np. funkcji lub klasy).

**iteracja** (ang. *iteration*) Wielokrotne wykonanie fragmentu kodu. Zobacz też **rekurencja**.

**iterator** (ang. *iterator*) obiekt stanowiący identyfikator elementu sekwencji.

**jednostka** (ang. *unit*) (1) Standardowa miara nadająca znaczenie wartościom (np. km oznacza odległość). (2) Dająca się wyodrębnić (np. nazwana) część całości.

**język programowania** (ang. *programming language*) Język służący do pisania programów.

**klasa** (ang. *class*) Zdefiniowany przez użytkownika typ, który może zawierać dane i funkcje składowe oraz typy składowe.

**klasa abstrakcyjna** (ang. *abstract class*) Klasa, której nie można bezpośrednio użyć do tworzenia obiektów. Często służy jako definicja interfejsu do klas pochodnych. Aby uczynić klasę abstrakcyjną, należy w niej zdefiniować funkcję czysto wirtualną lub chroniony konstruktor.

**klasa bazowa** (ang. *base class*) Klasa stanowiąca podstawę hierarchii klas. Większość klas bazowych zawiera przynajmniej jedną funkcję wirtualną.

**klasa konkretna** (ang. *concrete class*) Klasa, która pozwala na tworzenie obiektów.

**klasa pochodna, klasa derywowana, podklasa** (ang. *derived class*) Klasa utworzona na podstawie przynajmniej jednej innej klasy.

**kod** (ang. *code*) Program lub część programu. Słowem tym określa się zarówno kod źródłowy, jak i obiektyowy.

**kod obiektowy** (ang. *object code*) Wynik działania kompilatora stanowiący dane wejściowe dla konsolidatora (który utworzy z tego kod wykonywalny).

**kod źródłowy** (ang. *source code*) Kod napisany przez programistę i z założenia nadający się do czytania przez innych programistów.

**kompilator** (ang. *compiler*) Program tłumaczący kod źródłowy na kod obiektowy.

**kompromis** (ang. *trade-off*) Wynik dostosowania się do wymogów kilku kryteriów projektowych i implementacyjnych.

**koncepcja** (ang. *concept*) Zestaw wymagań, najczęściej dotyczących argumentu szablonu.

**konsolidator, program łączący** (ang. *linker*) Program łączący pliki z kodem obiektowym i biblioteki w wykonywalny program.

**konstruktor** (ang. *constructor*) Operacja inicjalizująca (budująca) obiekt. Zazwyczaj konstruktory ustalają niezmienniki i często zajmują zasoby potrzebne do używania obiektu (które są później zazwyczaj zwalniane przez destruktor).

**kontener** (ang. *container*) Obiekt przechowujący elementy (inne obiekty).

**kopiowanie** (ang. *copy*) Operacja, w wyniku której dwa obiekty mają takie same wartości. Zobacz również **przenoszenie**.

**koszt** (ang. *cost*) Ilość zasobów (np. czasu programisty, czasu działania lub przestrzeni) potrzebnych do utworzenia lub wykonywania programu. Najlepiej, gdy koszt jest funkcją złożoności.

**liczba całkowita** (ang. *integer*) Liczba bez części ułamkowej, np. 42 czy -99.

**liczba zmiennoprzecinkowa** (ang. *floating-point number*) komputerowe przybliżenie liczb rzeczywistych, np. 7.93 i 10.78e-3.

**literał** (ang. *literal*) Zapis bezpośrednio wyrażający jakąś wartość, np. 12 oznacza wartość „dwanaście”.

**łańcuch** (ang. *string*) Sekwencja znaków.

**nadtyp** (ang. *supertype*) Typ bazowy. Typ będący nadzbiorem innego typu.

**nagłówek** (ang. *header*) Plik zawierający deklaracje używane w różnych częściach programu.

**niezainicjowany** (ang. *uninitialized*) Niezdefiniowany stan obiektu przed jego inicjalizacją.

**niezmiennik** (ang. *invariant*) Coś, co zawsze w danym punkcie (lub punktach) programu musi być prawdziwe. Zazwyczaj służy do opisu stanu (zestawu wartości) obiektu lub pętli przed rozpoczęciem powtarzanej instrukcji.

**obcięcie** (ang. *truncation*) Strata informacji w wyniku konwersji jednego typu na taki, który nie może precyzyjnie odtworzyć konwertowanej wartości.

**obiekt** (ang. *object*) (1) Zainicjowany obszar pamięci o określonym typie, który przechowuje wartość tego typu. (2) Obszar pamięci.

**operacja** (ang. *operation*) Coś, co może wykonywać działania, np. funkcja lub operator.

**oprogramowanie** (ang. *software*) Zbiór fragmentów kodu i związanych z nimi danych. Słowo to często jest stosowane zamiennie ze słowem **program**.

**paradygmat** (ang. *paradigm*) Nieco pretensjonalnie brzmiące określenie projektu lub stylu programowania. Często słowo to jest błędnie używane w kontekstach implikujących, że istnieje jakiś paradygmat, który jest lepszy od wszystkich innych.

**parametr** (ang. *parameter*) Deklaracja danych przekazywanych funkcji lub szablonowi. Wywołana funkcja uzyskuje dostęp do przekazanych jej argumentów poprzez nazwy parametrów.

**pełzanie funkcji** (ang. *feature creep*) Tendencja do dodawania do programu zbyt wielu funkcji „tak na wszelki wypadek”.

**pętla** (ang. *loop*) Fragment kodu, który jest wykonywany wielokrotnie. W języku C++ jest to zwykle instrukcja `for` lub `while`.

**pętla nieskończona** (ang. *infinite loop*) Pętla, której warunek zakończenia nigdy nie będzie spełniony. Zobacz też **iteracja**.

**plik** (ang. *file*) Trwały zbiornik informacji w komputerze.

**plik obiektowy** (ang. *object file*) Plik zawierający kod obiektowy.

**plik źródłowy** (ang. *source file*) Plik zawierający kod źródłowy.

**podtyp** (ang. *subtype*) Typ pochodny. Typ, który ma wszystkie własności innego typu i potencjalnie dodatkowe inne.

**poprawność** (ang. *correctness*) Program lub część programu są poprawne, jeśli spełniają wymagania specyfikacji. Niestety specyfikacje bywają niepełne lub niespójne albo nie odpowiadają oczekiwaniom użytkowników. Dlatego, aby napisać kod do przyjęcia, często trzeba zrobić dużo więcej niż tylko trzymać się formalnej specyfikacji.

**pozyskanie zasobów oznacza inicjalizację** (ang. *Resource Acquisition Is Initialization — RAII*) Podstawowa technika zarządzania zasobami w oparciu o zakresy dostępności.

**program** (ang. *program*) Kod (zwykle z zestawem danych), który można uruchomić na komputerze.

**programowanie** (ang. *programming*) Sztuka wyrażania rozwiązań problemów w postaci kodu.

**programowanie obiektowe** (ang. *object-oriented programming*) Styl programowania oparty na wykorzystaniu klas i hierarchii klas.

**programowanie ogólne** (ang. *generic programming*) Styl programowania koncentrujący się na projektowaniu wydajnych implementacji algorytmów. Algorytmy ogólne działają na wszystkich typach argumentów, które spełniają ich wymagania. W języku C++ programowanie ogólne zwykle opiera się na wykorzystaniu szablonów.

**projekt** (ang. *design*) Ogólny opis wymaganego przez specyfikację sposobu działania programu.

**przeciążenie** (ang. *overload*) Zdefiniowanie dwóch funkcji lub operatorów o takich samych nazwach, ale przyjmujących różne argumenty.

**przedział** (ang. *range*) Sekwencja wartości z określonym początkiem i końcem. Na przykład [0,5) oznacza wartości 0, 1, 2, 3, 4.

**przenoszenie** (ang. *move*) Operacja polegająca na przeniesieniu wartości z jednego obiektu do innego i pozostawieniu wartości reprezentującej „pustkę”. Zobacz również **kopiowanie**.

**przepelnienie** (ang. *overflow*) Wytworzenie wartości, której nie da się zapisać w przeznaczonym dla niej miejscu.

**przesłanianie** (ang. *override*) Zdefiniowanie w klasie pochodnej funkcji o takiej samej nazwie i typach argumentów jak funkcja wirtualna w klasie nadrzędnej. Dzięki temu funkcja z podklasy może być wywoływana poprzez interfejs klasy bazowej.

**przetwarzanie** (ang. *computation*) Wykonywanie kodu połączone zwykle z pobieraniem danych i zwracaniem wyników.

**przybliżenie** (ang. *approximation*) Coś (np. wartość lub projekt) bliskiego ideałowi. Przybliżenie często jest wynikiem kompromisów.

**przypadek użycia** (ang. *use case*) Specyficzny (zwykle prosty) sposób użycia programu mający na celu sprawdzić jego funkcjonowanie i przeznaczenie.

**pseudokod** (ang. *pseudo code*) Opis procedury zapisany za pomocą nieformalnej notacji zamiast w jakimś języku programowania.

**referencja** (ang. *reference*) (1) Wartość określająca miejsce występowania w pamięci wartości określonego typu. (2) Zmienna przechowująca taką wartość.

**rekurencja** (ang. *recursion*) Wywołanie funkcji przez samą siebie. Zobacz też **iteracja**.

**rekurencja nieskończona** (ang. *infinte recursion*) Rekurencja, którą przerywa dopiero wyczerpanie pamięci do przechowywania stosu wywołań. W rzeczywistości taka rekurencja nie jest nieskończona, ponieważ zakończy ją jakiś błąd sprzętu.

**sekwencja** (ang. *sequence*) Zbiór elementów, który można przeglądać w porządku liniowym.

**słowo** (ang. *word*) Podstawowa jednostka pamięci w komputerze. Zwykle służy do przechowywania liczb całkowitych.

**specyfikacja** (ang. *specification*) Opis pożądanego sposobu działania kodu.

**stała** (ang. *constant*) Wartość, której nie można zmieniać (w danym zakresie). Wartość niezmienna.

**stan** (ang. *state*) Zbiór wartości.

**standard** (ang. *standard*) Oficjalnie uzgodniona definicja, np. języka programowania.

**styl** (ang. *style*) Zbiór technik programistycznych, których stosowanie pozwala uzyskać spójny efekt wykorzystania własności języka. Czasami słowem tym określa się tylko niskopoziomowe zasady dotyczące nazywania i wyglądu kodu.

**system** (ang. *system*) (1) Program lub zbiór programów wykonujących w komputerze określone zadania. (2) Skrót określenia „system operacyjny”, tzn. podstawowe środowisko wykonawcze i zestaw narzędzi komputera.

**szablon** (ang. *template*) Klasa lub funkcja parametryzowana przynajmniej jednym typem lub (w czasie kompilacji) wartością. W języku C++ szablony stanowią podstawę programowania ogólnego.

**tablica** (ang. *array*) Jednolita sekwencja elementów, zwykle ponumerowanych (np. [0,max)).

**testowanie** (ang. *testing*) Systematyczne poszukiwanie błędów w programie.

**typ** (ang. *type*) Konstrukcja definiująca zestaw możliwych wartości i zbiór operacji obiektu.

**uchwyt** (ang. *handle*) Klasa umożliwiająca dostęp do innej klasy poprzez składową będącą wskaźnikiem lub referencją. Zobacz również **kopiowanie**, **przenoszenie**, **zasób**.

**ukrywanie** (ang. *hiding*) Czynności mające na celu uniemożliwienie dostępu lub zobaczenia informacji. Na przykład nazwa z zagnieżdżonego (wewnętrznego) zakresu może uniemożliwić bezpośrednie używanie takiej samej nazwy z zakresu zewnętrznego.

**ukrywanie informacji** (ang. *information hiding*) Oddzielenie interfejsu od implementacji, czego celem jest ukrycie szczegółów implementacyjnych, które nie są przeznaczone dla użytkownika, i utworzenie warstwy abstrakcji.

**wartość** (ang. *value*) Zbiór bitów w pamięci interpretowanych zgodnie z ich typem.

**warunek końcowy** (ang. *post-condition*) Warunek, który musi być spełniony w chwili kończenia wykonywania określonego fragmentu kodu, np. funkcji lub pętli.

**warunek wstępny** (ang. *pre-condition*) Warunek, który musi być spełniony na początku wykonywania określonego fragmentu kodu, np. funkcji lub pętli.

**właściciel** (ang. *owner*) Obiekt odpowiedzialny za zwolnienie zasobu.

**wskaźnik** (ang. *pointer*) (1) Wartość identyfikująca w pamięci obiekt określonego typu. (2) Zmienna przechowująca taką wartość.

**wykonywalny** (ang. *executable*) Program gotowy do działania na komputerze.

**wymóg** (ang. *requirement*) (1) Opis pożądanego zachowania programu lub jego części. (2) Opis założeń przyjętych w funkcji lub szablonie na temat odbieranych przez nie argumentów.

**wyrażenie regularne** (ang. *regular expression*) Notacja do opisu wzorców w łańcuchach znaków.

**zakres** (ang. *scope*) Obszar tekstu programu (kodu źródłowego), w którym można używać określonej nazwy.

**zaokrąglanie** (ang. *rounding*) Konwersja wartości na wartość mniej precyzyjnego typu najlepiej jej odpowiadającą matematycznie.

**zasób** (ang. *resource*) Coś, co można zająć i trzeba później zwolnić, np. uchwyt do pliku, blokada czy pamięć. Zobacz również **uchwyt**, **właściciel**.

**złożoność** (ang. *complexity*) Trudne do precyzyjnego zdefiniowania pojęcie oznaczające poziom trudności opracowania rozwiązania problemu. Czasami słowem **złożoność** określa się szacowaną liczbę operacji potrzebnych do wykonania algorytmu.

**zmienna** (ang. *variable*) Nazwany obiekt określonego typu. Jeśli jest zainicjowana, zawiera jakąś wartość.

**zmienny** (ang. *mutable*) Taki, który można zmienić. Przeciwnieństwo określeń „niezmienny” i „stały”.