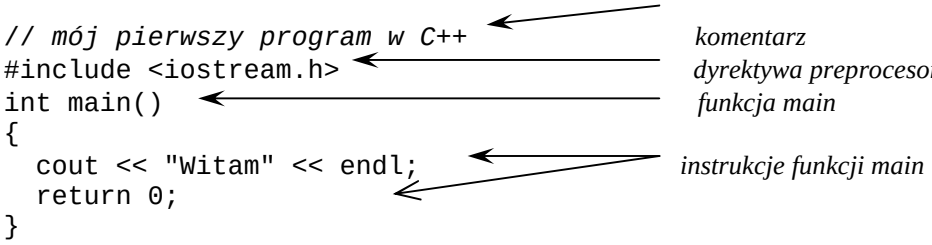
Program w języku C++



Komentarz

Znaki `//` oznaczają początek komentarza. Komentarz rozciąga się od znaków `//` do końca wiersza.

Znaki `//` pozwalają umieścić w programie komentarz jednowierszowy. Jeśli komentarz ma obejmować kilka wierszy, należy umieścić go pomiędzy znakami `/*` oraz `*/` , na przykład:

```
/* Autor: Jan Kowalski
   Data modyfikacji: 2.03.2001
*/
```

Kompilator ignoruje komentarze; zadaniem komentarza jest bowiem wyjaśnienie programu człowiekowi.

Dyrektywa `#include`

W języku C++ (za językiem C) przyjęto, że wprowadzanie danych i wyprowadzanie wyników będą realizowane za pomocą standardowej biblioteki. Biblioteka ta jest dołączana podczas kompilacji programu. W programie należy umieścić dyrektywę `#include`, która określa tę część biblioteki standardowej, z której będziemy korzystać:

```
#include <nazwa_pliku_nagłówkowego_do_włączenia>
```

Dla wejścia-wyjścia w stylu C++ będzie to plik nagłówkowy (ang. *header file*) `iostream.h`.

Funkcja `main`

Funkcja jest to element składowy programu, który ma swoją nazwę i który może być wywoływany czyli uruchamiany (ang. *call*) z innej części programu. Każdy program w języku C++ posiada funkcję o nazwie `main`. Od niej rozpoczyna się wykonywanie programu. Funkcja składa się z instrukcji, z których każda jest zakończona średnikiem.

Zapis `int main()`oznacza, że funkcja po wykonaniu zwraca wartość całkowitą (ang. *integer*), na podstawie której system operacyjny może stwierdzić, czy program zakończył się prawidłowo. Nawiasy `()` sygnalizują kompilatorowi, że nazwa `main` jest funkcją.

Nawiasy klamrowe `{}` oznaczają w języku C++ jakąś jednostkę. W tym przypadku oznaczają, że wszystkie instrukcje w nich zawarte należą do funkcji `main`.

Korzystanie ze standardowej biblioteki do wyprowadzania wyników

Instrukcja, która wyświetla tekst na ekranie ma postać:

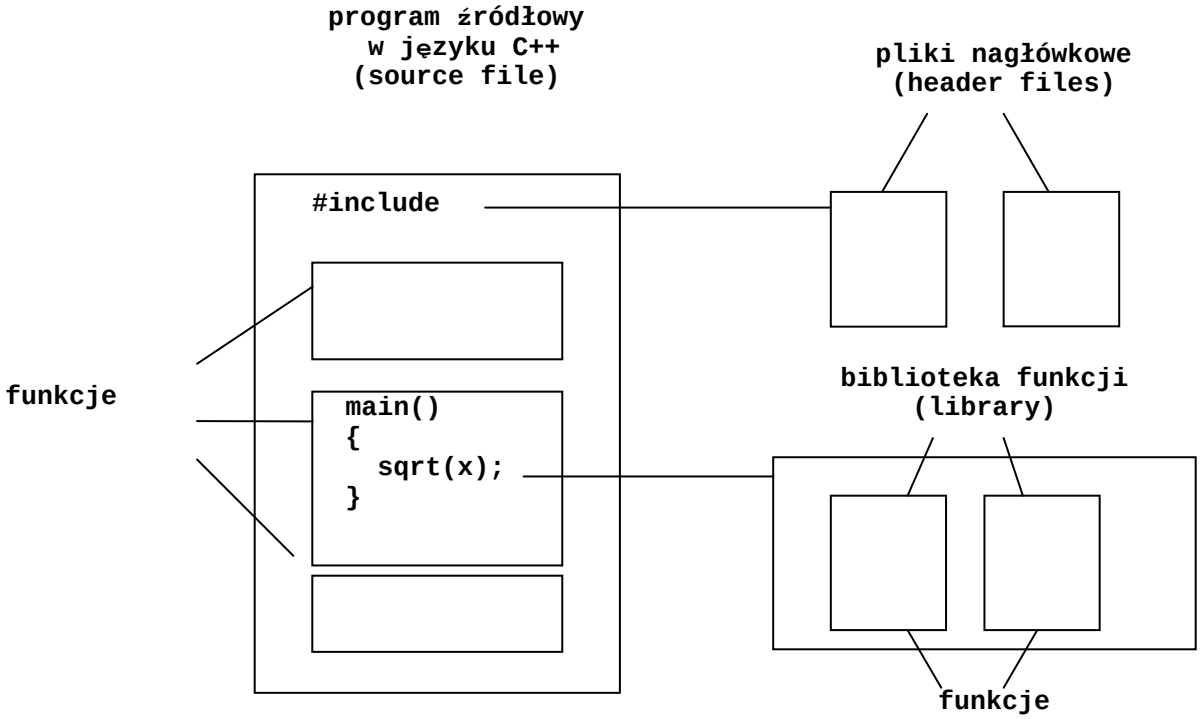
```
cout << "Witam" << endl;
```

Nazwa `cout` oznacza standardowy strumień wyjściowy (ang. *standard output stream*), który domyślnie jest powiązany z ekranem. Operator `<<` oznacza przesyłanie do strumienia wyjściowego. W tym przypadku przesyłany jest tekst oraz manipulator `endl`. Wysłanie manipulatora do strumienia wyjściowego oznacza dokonanie jakiegoś przekształcenia, w rzypadku `endl` jest to przejście do następnego wiersza.

Instrukcja `return`

Instrukcja `return` kończy wykonywanie funkcji i przekazuje określoną wartość do programu (lub innej funkcji), z którego funkcja ta została wywołana. Typ zwracanej wartości musi odpowiadać typowi, który został podany w nagłówku funkcji. Funkcja `main()` zwraca wartość systemowi operacyjnemu. Przyjmuje się, że 0 oznacza prawidłowe zakończenie programu.

Struktura programu w języku C++



Program w języku C++

- składa się z jednego lub wielu plików, w każdym pliku znajduje się jedna lub wiele funkcji
- zawsze musi zawierać funkcję o nazwie **`main()`**, od której rozpoczyna się wykonywanie programu
- może zawierać odwołania do plików nagłówkowych i funkcji bibliotecznych
- nazwy potrzebnych plików nagłówkowych wskazywane są za pomocą dyrektywy **`#include`**
- nazwy bibliotek wskazywane są za pomocą parametrów kompilacji

Pliki nagłówkowe

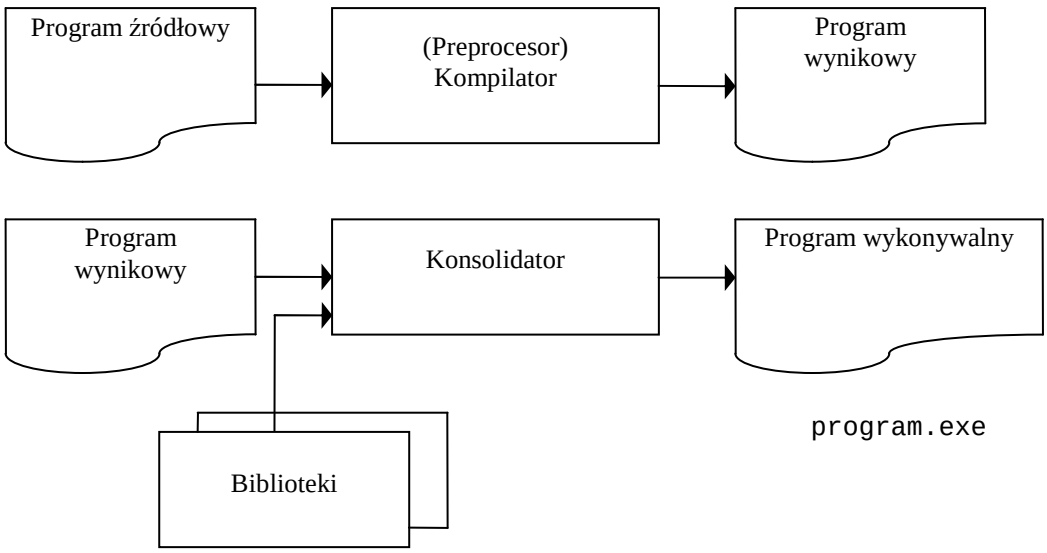
- zawierają deklaracje i definicje dla wszystkich funkcji, z których korzysta program i które nie znajdują się bezpośrednio w podstawowym pliku z tekstem programu
- są to pliki *tekstowe*
- nazwy plików nagłówkowych mają rozszerzenie **`.h`**
- przykład: **`iostream.h`**, **`math.h`**

Biblioteki funkcji

- zawierają funkcje włączane do programu podczas konsolidowania (linkowania)
- każda biblioteka ma swój plik nagłówkowy (jeden lub wiele)
- są dostarczane przez producenta kompilatora (np. biblioteka *run-time - czasu wykonania*) lub tworzone przez użytkownika
- są to pliki *binarne*
- mogą mieć różne rozszerzenia, na przykład **`.lib`**

Kompilacja programu

`program.cpp`

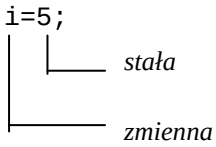


- Program źródłowy (ang. *source code*): dający się odczytać tekst programu.
- Kod wynikowy (ang. *object code*): przekład tekstu źródłowego programu na język komputerowy.
- Program wykonywalny (ang. *executable program*): program binarny gotowy do wykonania
- Kompilator (ang. *compiler*): program, który wczytuje cały tekst źródłowego i przekłada go na język komputerowy
- Konsolidator (ang. *linker*): program, który łączy oddzielnie skompilowane funkcje w jeden program; wiąże on kod wynikowy z funkcjami biblioteki C++
- Biblioteka (ang. *library*): plik zawierający funkcje standardowe, z których można korzystać w programie. Podczas wywołania takiej funkcji kompilator zapamiętuje jej nazwę, zaś konsolidator łączy kod wynikowy tekstu źródłowego z kodem istniejącym w bibliotece.

Typy danych

Zmienne i stałe

- Dane programu zapisywane są w postaci *zmiennych* lub *stałych*.
- *Zmienna* (ang. *variable*) to pewien obszar pamięci o nadanej symbolicznej nazwie, w którym można przechowywać dane, pobierać je i zmieniać podczas wykonywania programu. Zmienną nazywamy inaczej *obiektem*.
- *Stała* (*literał*, ang. *literal constant*) to dana zapisana w sposób dosłowny (liczba, znak, napis), nie zmienia się podczas wykonywania programu.
- Przykład:



Identyfikatory czyli nazwy

- *Identyfikator* (ang. *identifier*) to nazwa służąca do oznaczania obiektów zdefiniowanych przez użytkownika w programie (zmiennych, funkcji, nazwanych stałych).
- Nazwa może zawierać litery, cyfry, znak podkreślenia, przy czym może rozpoczynać się od litery lub znaku podkreślenia.
- Rozróżniane są małe i wielkie litery. Oznacza to, że x oraz X to dwie różne zmienne.
- Nazwa nie może być słowem kluczowym języka C++ (np. nazwą instrukcji `if`) ani być używana przez bibliotekę standardową.
- Standard ANSI C ogranicza rozmiar nazw do 31 znaków. Standard C++ ANSI/ISO nie nakłada ograniczeń na długość nazw. Jednakże większość kompilatorów przyjmuje swoje maksymalne długości nazw określane w opcjach kompilatora.
- Przykłady:

Poprawne nazwy	Niepoprawne nazwy
`licznik_12`	`12_licznik`
`test12`	`test-12`
`test_12`	`test..12`

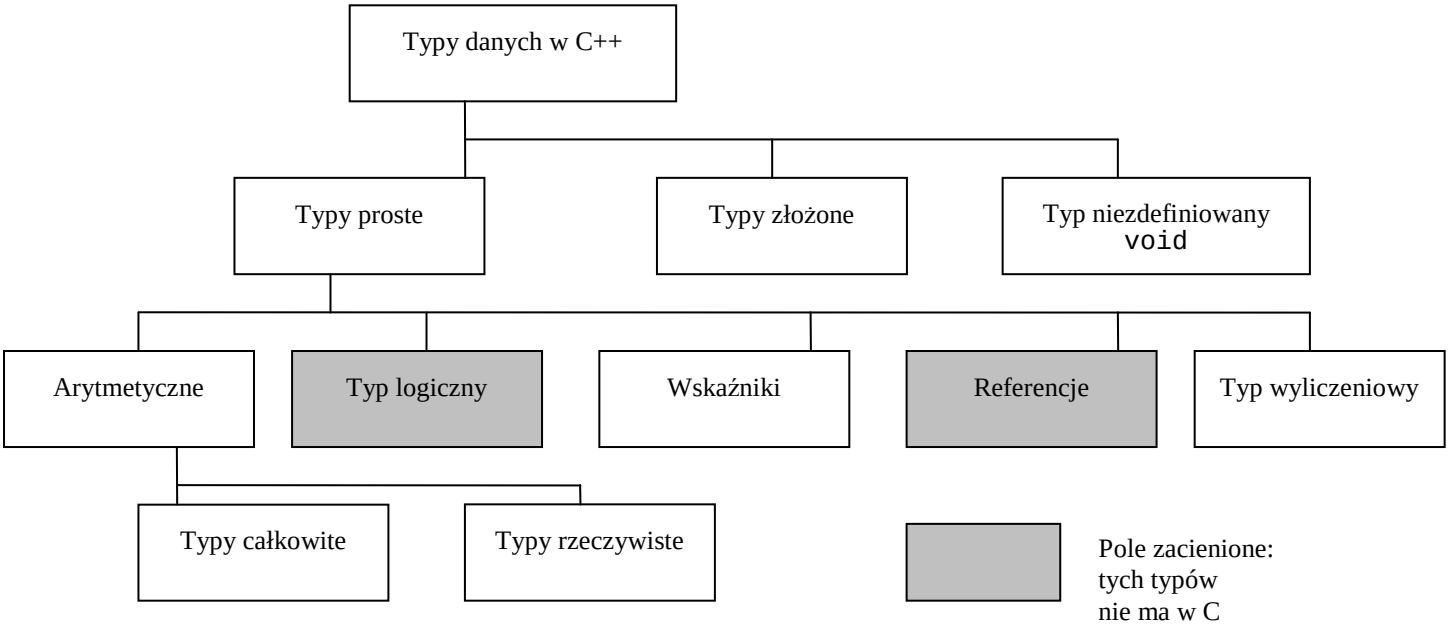
- Ustalając nazwy dobrze jest stosować kilka ogólnie przyjętych zasad:
  - ***powinno to być słowo kojarzące się z przeznaczeniem zmiennej***
  - najczęściej używa się małych liter, np. indeks nie zaś INDEKS
  - jeśli nazwa składa się z kilku słów, poszczególne słowa oddziela się znakiem podkreślenia, albo zaczyna się każde słowo od wielkiej litery oprócz pierwszego
  - wybrany styl tworzenia nazw powinien być jednorodny w całym programie

Słowa kluczowe

- W języku C++ słowa o zastrzeżonym znaczeniu nazywane są *słowami kluczowymi* (ang. *keywords*). Jest ich 63.
- Tych słów nie można używać jako nazwy zmiennych i innych obiektów.
- Przykłady słów kluczowych:
  - `int` - używane jest do określenia typu całkowitego
  - `if` - instrukcja wyboru
  - `sizeof` – operator zwracający rozmiar argumentu w bajtach

Podstawowe typy danych

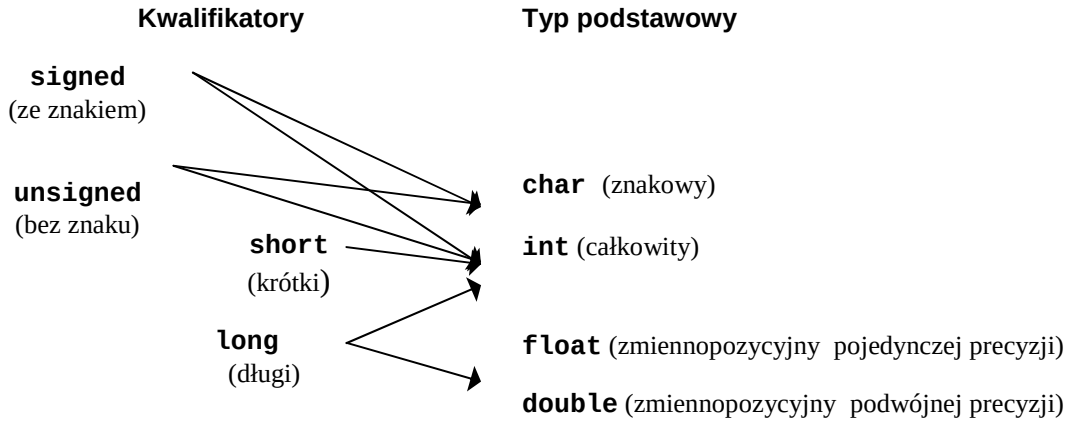
- Dane (zmienne, stałe, wartości wyrażeń, wartości generowane przez funkcje) przechowywane są w pamięci jako ciąg bitów.
- *Typ danych* - nadaje znaczenie ciągowi bitów o określonym adresie i długości:
    - określa ile pamięci potrzeba do przechowania danej,
    - jakie operacje mogą być wykonane na danej,
    - jak te operacje są interpretowane.



- Zbiór typów prostych dostępnych w języku C++ i C jest podobny.
- Typy *proste* (skalarne, ang. *scalar types*) są niepodzielne, wykorzystuje się je do przechowywania pojedynczych danych.
    - *typy arytmetyczne*: do przechowywania liczb całkowitych, rzeczywistych, znaków,
    - *typ logiczny*: do przechowywania wartości logicznych *prawda* i *fałsz* (nie ma go w C)
    - *typ wskaźnikowy*: do przechowywania adresów obiektów danego typu
    - *typ referencyjny*: pozwala nadać inną nazwę obiektowi (nie ma go w C)
    - *typ wyliczeniowy*: do reprezentowania zbioru wartości podanych przez użytkownika.
- Typy *złożone* (ang. *aggregate types*): składają się z elementów typów prostych oraz innych typów złożonych. Wykorzystywane do przechowywania danych powiązanych ze sobą. Do typów złożonych należą:
    - *tablice* obiektów danego typu,
    - *struktury* (i *klasy*) zawierające zestawy obiektów różnego typu,
    - *unie* zawierające dowolny z zestawu obiektów o różnych typach,
    - *funkcje* zwracające wartości danego typu.
- Typ `void`: typ ten ma specjalne zastosowania, nie ma obiektów typu `void`; używany jest na przykład wtedy, kiedy chcemy powiedzieć, że funkcja nie zwraca żadnej wartości.

Typy arytmetyczne

- Typy arytmetyczne służą do przechowywania liczb całkowitych, rzeczywistych i znaków.
- Są one dostępne w różnych rozmiarach, dzięki czemu programista może wybrać ilość zużywanej pamięci, precyzję i zakres przechowywanych liczb.



- Rozmiary i zakresy danych poszczególnych typów mogą być różne dla różnych implementacji.
- Dla typów `int` i `char` określana jest najmniejsza i największa wartość liczby, która może być przechowywana w zmiennej tego typu.
- Dla typów `float` i `double` określane są:
  - największa liczba możliwa do przedstawienia
  - najmniejsza liczba możliwa do przedstawienia
  - rozróżnialność - najmniejsza dodatnia liczba x taka, że $1.0+x \neq 1.0$
  - dokładność - liczba cyfr dziesiętnych, które mogą być przedstawione dokładnie
- Największe i najmniejsze wartości każdego typu w danej implementacji są podane w standardowych plikach nagłówkowych: `limits.h` i `float.h`.

- Niektóre nazwy typów można podawać w postaci pełnej i skróconej. Na przykład mówiąc o typie `signed int` najczęściej pomija się kwalifikator i stosuje się nazwę krótszą `int`.

Długi format nazwy typu	Krótki format nazwy typu
`char`	`char`
`signed char`	`signed char`
`unsigned char`	`unsigned char`
`signed short int`	**`short`**
`unsigned short int`	**`unsigned short`**
`signed int`	**`int`** (lub **`signed`**)
`unsigned int`	**`unsigned`**
`signed long int`	**`long`**
`unsigned long int`	**`unsigned long`**
`float`	`float`
`double`	`double`
`long double`	`long double`

- Standard C++ ANSI/ISO określa tylko minimum jakie musi spełniać dany typ:

Nazwa typu	Zakresy przechowywanych wartości (standard ANSI/ISO)	Do czego służy
Typy całkowite		
`char`	Musi być wystarczająco duży, aby można było przechować w nim dowolny znak ze zbioru wspieranego przez implementację, np. 127 znaków zbioru podstawowego ASCII lub 255 znaków zbioru rozszerzonego ASCII. Zmienna typu char przyjmuje wartości takie jak `signed char` lub `unsigned char` w zależności od implementacji.	bardzo małe liczby całkowite, kody znaków, np. ASCII
`signed char`	-127 do 127	bardzo małe liczby całkowite, kody znaków, np. ASCII
`unsigned char`	0 do 255	bardzo małe liczby całkowite dodatnie, kody ASCII
`int`	Obiekty typu `int` zajmują jedno słowo; Jeśli w danym systemie operacyjnym przyjęte jest słowo 16 bitowe oznacza to liczby z zakresu --32 768 do 32 767; Jeśli zaś słowo wynosi 32 bity, zakres liczb wynosi od -2 147 483 648 do 2 147 483 647	średnie lub duże liczby całkowite
`short int`	Co najmniej połowę słowa; jednak w komputerach 16 bitowych przyjęte jest, że typ `short` i `int` mają ten sam rozmiar	średnie liczby całkowite
`long int`	Co najmniej tyle co `int`; w komputerach 16 bitowych przyjęte jest używać dwa słowa, w komputerach 32 bitowych zazwyczaj `int` i `long int` mają te same rozmiary	duże liczby całkowite
`unsigned int`	Taki sam rozmiar jak dla `int`. Dla `int` 16 bitowego oznacza to wartości od 0 do 65 535 ; zaś dla `int` 32 bitowego od 0 do 4 294 967 295	średnie lub większe liczby całkowite dodatnie
`unsigned short int`	Taki sam rozmiar co `short int`; Dla `short` 16 bitowego oznacza to wartości od 0 do 65 535.	średnie liczby całkowite dodatnie
`unsigned long int`	Taki sam rozmiar co `long int`; Dla `long int` 32 bitowego oznacza to wartości od 0 do 4 294 967 295	bardzo duże liczby całkowite dodatnie
Typy rzeczywiste		
`float`	zależy od implementacji, zbiór wartości `float` stanowi podzbiór wartości `double`	liczby rzeczywiste przedstawione z pojedynczą dokładnością – przykładowo małe liczby rzeczywiste o 7 cyfrach dokładności
`double`	zawiera co najmniej zbiór wartości `float`, co najmniej taka dokładność jak `float`,	liczby rzeczywiste przedstawione z podwójną dokładnością – przykładowo duże liczby rzeczywiste o 15 cyfrach dokładności
`long double`	zawiera zbiór wartości `double`, co najmniej taka dokładność jak `double`,	bardzo duże liczby rzeczywiste (18 cyfr dokładności)

- Przykład: implementacja typów w kompilatorze Borland C++ - wersja DOS

Nazwa typu	Rozmiar (bity)	Zakres przechowywanych wartości	Do czego służy
całkowity		**liczba całkowita**	
`short int`	16	-32768 do 32767	średnie liczby całkowite
`int`	16	-32768 do 32767	średnie liczby całkowite
`long int`	32	-2 147 483 648 do 2 147 483 647	duże liczby całkowite
modyfikacje		**dodatnia liczba całkowita**	
`unsigned int`	16	0 do 65535	większe liczby całkowite dodatnie
`unsigned long int`	32	0 do 4 294 967 295	bardzo duże liczby całkowite dodatnie
zmiennopozycyjny		**liczba rzeczywista**	
`float`	32	$1{,}1*10^{-38}$ do $3{,}4*10^{38}$	małe liczby rzeczywiste (7 cyfr dokładności)
`double`	64	$2{,}2*10^{-308}$ do $1{,}7*10^{308}$	duże liczby rzeczywiste (15 cyfr dokładności)
`long double`	80	$3{,}3*10^{-4917}$ do $1{,}1*10^{4932}$	bardzo duże liczby rzeczywiste (18 cyfr dokładności)
znakowy (specjalny całkowity)		**znak lub mała liczba całkowita**	
`char`	8	-128 do 127	bardzo małe liczby całkowite, znaki ASCII
`signed char`	8	-128 do 127	
`unsigned char`	8	0 do 255	bardzo małe liczby całkowite dodatnie, pełen zestaw kodów PC

Uwaga:

1. Typ `char` w różnych kompilatorach może być interpretowany jako `signed char` lub `unsigned char`. Ustawienie to prawie zawsze da się zmienić.
2. Borland C++ domyślnie przyjmuje, że `char` oznacza `signed char`, o ile użytkownik nie zmieni ustawienia domyślnego.

Typ logiczny bool

- Typ logiczny służy do przechowywania wartości logicznych.
- Zmienne tego typu mogą przyjmować dwie wartości: `true` (prawda) i `false` (fałsz).
- Typ logiczny wprowadzono w języku C++. W języku C do wyrażenia wartości logicznych wykorzystywany jest typ `int`. Przyjęto następującą konwencję: 0 oznacza fałsz, wartość różna od zera oznacza prawdę.

Nadawanie typu zmiennym i stałym

- ***Typ i nazwę zmiennej (ang. variable) określa użytkownik jawnie za pomocą odpowiedniej deklaracji, biorąc pod uwagę dane, które mają być w zmiennej przechowywane.***
- Deklaracja typu składa się ze specyfikatora typu i występującej po niej nazwy zmiennej. Musi być zakończona średnikiem.
- Jeśli definiujemy kilka zmiennych tego samego typu można je umieścić w jednej deklaracji typu, oddzielając nazwy przecinkami.
- Przykłady:

```
int licznik; /* zmienna typu int o nazwie licznik
  (może przechowywać liczby całkowite z zakresu określonego typem int) */
int a,b,c;   /* trzy zmienne a, b, c tego samego typu */
double suma; /* zmienna typu double o nazwie suma
  (może przechowywać l. rzeczywiste z zakresu określonego typem double)*/
char znak;      /* zmienna typu char o nazwie znak
  (może przechowywać znaki lub małe liczby całkowite
   z zakresu określonego typem char)*/
bool flaga;     /* zmienna typu bool o nazwie flaga
  (może przechowywać wartość true lub false ) */
```

- Deklaracja typu musi poprzedzać w tekście programu użycie zmiennej.
- W języku C deklaracja typu mogła występować tylko na początku funkcji (programu lub bloku), przed instrukcjami związanymi z wykonywaniem programu.
- W języku C++ deklaracja typu może występować w dowolnym miejscu tekstu programu, byle przed użyciem zmiennej.

- ***Typ stałej (ang. literal constant) określa użytkownik odpowiednio zapisując daną w instrukcji.***

```
int licznik = 0;   // stała 0 typu int: liczba bez kropki dziesiętnej
double suma = 0.;  // stała 0 typu double: liczba z kropką dziesiętną
char c='*';        // stała * typu char: znak z apostrofach
bool b=false;      // stała false typu bool: jedna z dwóch wartości false
                                                        // lub true
```

Nadawanie wartości zmiennym

- Zmienna przyjmuje wartości w wyniku *inicjowania*, *przypisania* lub *wczytania*.

- ***Inicjowanie***: jest to nadanie pierwszej wartości zmiennej podczas jej definiowania, nie jest obowiązkowe:

```
int ile,liczba=20;  /* ile nie ma wartości początkowej */
float powierzchnia=4.5;
long odl_od_Ksiezyca=238857;
```

- Zmienna, której nie nadano wartości początkowej to zmienna *niezainicjowana* (ang. *uninitialized*).
- Zmienna *zdefiniowana niezainicjowana* ma pewną wartość, ale wartość ta jest nieokreślona - wynik tego, że podczas rezerwowania obszaru pamięci dla zmiennej, pamięć ta nie jest czyszczona (*Uwaga*: są wyjątki - pewne zmienne są automatycznie inicjowane z wartością 0, patrz: *Zasięg deklaracji i czas trwania obiektów*).

```
// Przykład błędnie działającego programu
#include <iostream.h>
int main()
{
  int k;
  cout << "Wartosc k: " << k << endl;
  return 0;
}
```

Program się skompilował i można było go uruchomić. Kompilator Borland C++ ostrzegł tylko: `Possible use of 'k' before definition.` Jednakże wynik wykonania programu jest przypadkowy:

```
Wartosc k: 12
```

- ***Przypisanie***: w trakcie działania programu zmiennej przypisuje się wartość za pomocą operatora przypisania = :

```
int ile;
float powierzchnia;
ile=20;               // przypisanie
powierzchnia=4.5;     // przypisanie
```

- ***Wczytanie***: w trakcie działania programu można wczytywać wartości do zmiennych na przykład za pomocą konstrukcji `cin >> nazwa_zmiennej`:

```
int ile;
cin >> ile;
```