



Podstawy C++

TABLICE INFORMATYCZNE • Radosław Sokół



WPROWADZENIE

Język C++ stanowi rozszerzenie języka C. Prawie każdy program napisany zgodnie z zasadami języka C jest jednocześnie poprawnym programem C++ (choć niewykorzystującym możliwości tego języka).

Typy danych

Podstawowe typy danych są identyczne w językach C i C++. Jedynym nowym typem jest typ logiczny `bool`, obsługiwany standardowo (w języku C występuje on dopiero od wersji C99 i wymaga dołączenia pliku nagłówkowego `stdbool.h`).

Zapis	Alternatywny zapis	Znaczenie
<code>void</code>		brak wartości
<code>char</code>		liczba 8-bitowa, naturalna lub całkowita, albo znak alfanumeryczny
<code>short</code>	<code>short int</code>	liczba 8- lub 16-bitowa, naturalna lub całkowita
<code>int</code>		liczba 16- lub 32-bitowa, naturalna lub całkowita
<code>long</code>	<code>long int</code>	liczba 32- lub 64-bitowa, naturalna lub całkowita
<code>long long</code>	<code>long long int</code>	liczba 64-bitowa, naturalna lub całkowita (typ dostępny od wersji języka C++11)
<code>float</code>		liczba zmiennoprzecinkowa, typowo 32-bitowa
<code>double</code>		liczba zmiennoprzecinkowa, typowo 64-bitowa
<code>long double</code>		liczba zmiennoprzecinkowa, typowo 80-bitowa
<code>bool</code>		wartość logiczna (<code>true/false</code>)

Od wersji języka C++11 typ danych może zostać zastąpiony słowem kluczowym `auto`. W takim przypadku kompilator sam określa optymalny typ danych na podstawie wyrażenia znajdującego się po prawej stronie instrukcji przypisania: `std::vector<int> elements;`

```
...
auto i = elements.begin();
Wyrażenie decltype(x) zwraca w czasie kompilacji typ określony dla wyrażenia x:
int value = 10;
decltype(value) other_value = value;
```

Modyfikatory

Nazwa	Znaczenie
auto	Zmienna lokowana na stosie mikroprocesora; atrybut domyślny, stosowany przy braku innych.
register	Zmienna lokowana w rejestrze mikroprocesora; podnosi szybkość operowania na zawartości zmiennej; liczba rejestrów jest ograniczona, należy stosować w bardzo ograniczonym zakresie; kompilator ma prawo traktować to wyłącznie jako sugestię i pominąć w razie braku możliwości wykorzystania rejestru mikroprocesora.
volatile	Wyłączenie możliwości optymalizowania dostępu do zawartości zmiennej; atrybut wykorzystywany w stosunku do zmiennych reprezentujących rejestry sprzętowe urządzeń lub wykorzystywanych w środowisku współbieżnym.
const	Blokada możliwości zmiany zawartości zmiennej po jej początkowym nadaniu.
constexpr	Oznacza wyrażenia, których wartość może być wyznaczona w czasie kompilacji programu i nie ulega nigdy zmianie.
thread_local	Oznacza elementy na poziomie przestrzeni nazw lub statyczne elementy klas, które mają posiadać odrębną instancję dla każdego realizowanego wątku.

Literały

Literały liczbowe

Od wersji języka C++14 literały liczbowe mogą być podawane w formie binarnej z wykorzystaniem przedrostka `0b` lub `0B`:

```
auto a = 0b111010011010;
```

Dodatkowo od tej samej wersji języka literały liczbowe mogą zawierać znak `'` dowolnie separujący grupy cyfr. Może to zwiększać czytelność literałów o dużych wartościach. Prawidłowość podziału na grupy nie jest w żaden sposób weryfikowana i obydwa zapisy są poprawne:

```
auto a = 1'234'567;
```

```
auto b = 12'34'56'78;
```

Literały tekstowe Unicode

Od wersji języka C++11 literały tekstowe mogą być poprzedzone nowymi przedrostkami ponad obsługiwany wcześniej `L`:

Literał	Znaczenie
<code>"..."</code>	Tablica znaków <code>const char</code> (ASCII).
<code>L"..."</code>	Tablica znaków <code>wchar_t</code> (Unicode UCS-2 lub UCS-4).
<code>u8"..."</code>	Tablica znaków <code>const char</code> zawierająca tekst w kodowaniu Unicode UTF-8.
<code>u"..."</code>	Tablica znaków <code>const char16_t</code> zawierająca tekst w kodowaniu Unicode UTF-16.
<code>U"..."</code>	Tablica znaków <code>const char32_t</code> zawierająca tekst w kodowaniu Unicode UTF-32.

Od wersji C++11 literały tekstowe mogą być zapisywane z wyłączeniem trybu analizy treści za pomocą przedrostka `R`:

```
R"ogranicznik(.....)ogranicznik"
```

Ogranicznik może być dowolnym tekstem o długości nieprzekraczającej 16 znaków, w tym pustym:

```
R"(".....)"
```

Literały definiowane

Wersja języka C++11 wprowadziła możliwość definiowania własnych literałów:

```
typ_zwracany operator "" symbol(typ_wejsciuwy wejście);
```

```
typ_zwracany operator "" symbol(typ_wejsciuwy wejście, std::size_t rozmiar);
```

Definiują operator o symbolu `symbol`, pobierający pewne `wejście` i generujący wartość zwrótną tego samego lub innego typu. Typ zwracany jest dowolny. Typ wejściowy może należeć do ograniczonego zbioru wyznaczanego listą dopuszczalnych parametrów operatora definiującego literał:

<code>const char*</code> <code>unsigned long long int</code> <code>long double</code> <code>char</code> <code>wchar_t</code> <code>char16_t</code> <code>char32_t</code>	Pojedynczy parametr
<code>const char*</code> , <code>std::size_t</code> <code>const wchar_t*</code> , <code>std::size_t</code> <code>const char16_t*</code> , <code>std::size_t</code> <code>const char32_t*</code> , <code>std::size_t</code>	Dwa parametry: - wartość tekstowa - długość przekazanej wartości w znakach

Przykład:

```
double operator "" _km(const long double km) {
    return km * 1000.0;
}
double distance = 1.35_km;
```

Standardowe literały definiowane

Plik nagłówkowy <code><complex></code>
<code>constexpr std::complex<double> operator""i(long double arg);</code> <code>constexpr std::complex<double> operator""i(unsigned long long arg);</code> Zwraca liczbę zespoloną (<code>std::complex</code>) reprezentowaną przez wartości <code>double</code> , zawierające wyłącznie część urojoną.
<code>constexpr std::complex<float> operator""if(long double arg);</code> <code>constexpr std::complex<float> operator""if(unsigned long long arg);</code> Zwraca liczbę zespoloną (<code>std::complex</code>) reprezentowaną przez wartości <code>float</code> , zawierające wyłącznie część urojoną.
<code>constexpr std::complex<long double> operator""il(long double arg);</code> <code>constexpr std::complex<long double> operator""il(unsigned long long arg);</code> Zwraca liczbę zespoloną (<code>std::complex</code>) reprezentowaną przez wartości <code>long double</code> , zawierające wyłącznie część urojoną.
Plik nagłówkowy <code><chrono></code>
<code>constexpr std::chrono::hours operator""h(unsigned long long hrs);</code> <code>constexpr std::chrono::duration<typ, std::ratio<3600,1>> operator""h(long double hrs);</code> Konwertuje wartość liczbową na typ odpowiadający liczbie godzin (<code>std::chrono::hours</code>) lub ogólnie czasowi trwania odpowiadającemu godzinie (<code>std::chrono::duration</code>).
<code>constexpr std::chrono::minutes operator""min(unsigned long long mins);</code> <code>constexpr std::chrono::duration<typ, std::ratio<60,1>> operator""min(long double mins);</code> Konwertuje wartość liczbową na typ odpowiadający liczbie minut (<code>std::chrono::minutes</code>) lub ogólnie czasowi trwania odpowiadającemu minucie (<code>std::chrono::duration</code>).
<code>constexpr std::chrono::seconds operator""s(unsigned long long secs);</code> <code>constexpr std::chrono::duration<typ> operator""s(long double secs);</code> Konwertuje wartość liczbową na typ odpowiadający liczbie sekund (<code>std::chrono::seconds</code>) lub ogólnie czasowi trwania odpowiadającemu sekundzie (<code>std::chrono::duration</code>).
<code>constexpr std::chrono::milliseconds operator""ms(unsigned long long ms);</code> <code>constexpr std::chrono::duration<typ, std::milli> operator""ms(long double ms);</code> Konwertuje wartość liczbową na typ odpowiadający liczbie milisekund (<code>std::chrono::milliseconds</code>) lub ogólnie czasowi trwania odpowiadającemu milisekundzie (<code>std::chrono::duration</code>).
<code>constexpr std::chrono::microseconds operator""us(unsigned long long us);</code> <code>constexpr std::chrono::duration<typ, std::micro> operator""us(long double us);</code> Konwertuje wartość liczbową na typ odpowiadający liczbie mikrosekund (<code>std::chrono::microseconds</code>) lub ogólnie czasowi trwania odpowiadającemu mikrosekundzie (<code>std::chrono::duration</code>).
<code>constexpr std::chrono::nanoseconds operator""ns(unsigned long long nsec);</code> <code>constexpr std::chrono::duration<typ, std::nano> operator""ns(long double nsec);</code> Konwertuje wartość liczbową na typ odpowiadający liczbie nanosekund (<code>std::chrono::nanoseconds</code>) lub ogólnie czasowi trwania odpowiadającemu nanosekundzie (<code>std::chrono::duration</code>).
Plik nagłówkowy <code><string></code>
<code>std::string operator""s(const char *str, std::size_t len);</code> <code>std::u16string operator""s(const char16_t *str, std::size_t len);</code> <code>std::u32string operator""s(const char32_t *str, std::size_t len);</code> <code>std::wstring operator""s(const wchar_t *str, std::size_t len);</code> Tworzy obiekt <code>std::string</code> (lub inny, w zależności od rodzaju przekazywanego literału tekstowego) przechowujący dokładną zawartość źródłowego literału tekstowego. W szczególności źródłowy literał tekstowy może zawierać znaki o kodzie 0; nie będą one interpretowane jako koniec tekstu.
<code>constexpr std::string_view operator""sv(const char *str, std::size_t len) noexcept;</code> <code>constexpr std::u16string_view operator""sv(const char16_t *str, std::size_t len) noexcept;</code> <code>constexpr std::u32string_view operator""sv(const char32_t *str, std::size_t len) noexcept;</code> <code>constexpr std::wstring_view operator""sv(const wchar_t *str, std::size_t len) noexcept;</code> Dostępny od wersji języka C++17. Tworzy obiekt <code>std::string_view</code> (lub inny, w zależności od rodzaju przekazywanego literału tekstowego) bezpośrednio odwzorowujący podany literał tekstowy jako obiekt mający cechy obiektowej zmiennej tekstowej (patrz opis klasy <code>std::string_view</code>).

Typ wyliczeniowy

Zmienne typu `wyliczeniowego` mogą przyjmować wartości tylko z określonego zbioru.

```
enum Wyliczenie {
```

```
    wartość1,
```

```
    wartość2,
```

```
    ...
```

```
}
```

Wartości typu `wyliczeniowego` są wewnętrznie liczbami typu `int`. Pierwsza wartość typu `wyliczeniowego` otrzymuje numer 0. Możliwe jest określenie wprost liczb przypisanych do poszczególnych wartości. Wiele wartości typu `wyliczeniowego` może przyjmować te same wartości liczbowe:

```
enum Wyliczenie {
```

```
    wartość1,          // 0
```

```
    wartość2,          // 1
```

```
    wartość3 = 10,     // 10
```

```
    wartość4,          // 11
```

```
    wartość5 = 1,      // 1
```

```
    ...
```

```
}
```

*Dalsza część książki dostępna w wersji
pełnej.*

