Podstawy C++

TABLICE INFORMATYCZNE • Radosław Sokół

Helion

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

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 = 0b1110010011010;
```

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
`"..."`	Tablica znaków `const char` (ASCII).
`L"..."`	Tablica znaków `wchar_t` (Unicode UCS-2 lub UCS-4).
`u8"..."`	Tablica znaków `const char` zawierająca tekst w kodowaniu Unicode UTF-8.
`u"..."`	Tablica znaków `const char16_t` zawierająca tekst w kodowaniu Unicode UTF-16.
`U"..."`	Tablica znaków `const char32_t` 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_wejsciowy wejście);
typ_zwracany operator "" symbol(typ_wejsciowy wejście, std::size_t rozmiar);
```

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

`const char*`
`unsigned long long int`
`long double`
`char`
`wchar_t`
`char16_t`
`char32_t`
	Pojedynczy parametr

`const char*, std::size_t`
`const wchar_t*, std::size_t`
`const char16_t*, std::size_t`
`const char32_t*, std::size_t`
	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 `<complex>`

```
constexpr std::complex<double> operator""i(long double arg);
constexpr std::complex<double> operator""i(unsigned long long arg);
```
Zwraca liczbę zespoloną (`std::complex`) reprezentowaną przez wartości `double`, zawierające wyłącznie część urojoną.

```
constexpr std::complex<float> operator""if(long double arg);
constexpr std::complex<float> operator""if(unsigned long long arg);
```
Zwraca liczbę zespoloną (`std::complex`) reprezentowaną przez wartości `float`, zawierające wyłącznie część urojoną.

```
constexpr std::complex<long double> operator""il(long double arg);
constexpr std::complex<long double> operator""il(unsigned long long arg);
```
Zwraca liczbę zespoloną (`std::complex`) reprezentowaną przez wartości `long double`, zawierające wyłącznie część urojoną.

Plik nagłówkowy `<chrono>`

```
constexpr std::chrono::hours operator""h(unsigned long long hrs);
constexpr std::chrono::duration<typ, std::ratio<3600,1>> operator""h(long double hrs);
```
Konwertuje wartość liczbową na typ odpowiadający liczbie godzin (`chrono::hours`) lub ogólnie czasowi trwania odpowiadającemu godzinie (`chrono::duration`).

```
constexpr std::chrono::minutes operator""min(unsigned long long mins);
constexpr std::chrono::duration<typ, std::ratio<60,1>> operator""min(long double mins);
```
Konwertuje wartość liczbową na typ odpowiadający liczbie minut (`std::chrono::minutes`) lub ogólnie czasowi trwania odpowiadającemu minucie (`std::chrono::duration`).

```
constexpr std::chrono::seconds operator""s(unsigned long long secs);
constexpr std::chrono::duration<typ> operator""s(long double secs);
```
Konwertuje wartość liczbową na typ odpowiadający liczbie sekund (`std::chrono::seconds`) lub ogólnie czasowi trwania odpowiadającemu sekundzie (`std::chrono::duration`).

```
constexpr std::chrono::milliseconds operator""ms(unsigned long long ms);
constexpr std::chrono::duration<typ, std::milli> operator""ms(long double ms);
```
Konwertuje wartość liczbową na typ odpowiadający liczbie milisekund (`std::chrono::milliseconds`) lub ogólnie czasowi trwania odpowiadającemu milisekundzie (`std::chrono::duration`).

```
constexpr std::chrono::microseconds operator""us(unsigned long long us);
constexpr std::chrono::duration<typ, std::micro> operator""us(long double us);
```
Konwertuje wartość liczbową na typ odpowiadający liczbie mikrosekund (`std::chrono::microseconds`) lub ogólnie czasowi trwania odpowiadającemu mikrosekundzie (`std::chrono::duration`).

```
constexpr std::chrono::nanoseconds operator""ns(unsigned long long nsec);
constexpr std::chrono::duration<typ, std::nano> operator""ns(long double nsec);
```
Konwertuje wartość liczbową na typ odpowiadający liczbie nanosekund (`std::chrono::nanoseconds`) lub ogólnie czasowi trwania odpowiadającemu nanosekundzie (`std::chrono::duration`).

Plik nagłówkowy `<string>`

```
std::string operator""s(const char *str, std::size_t len);
std::u16string operator""s(const char16_t *str, std::size_t len);
std::u32string operator""s(const char32_t *str, std::size_t len);
std::wstring operator""s(const wchar_t *str, std::size_t len);
```
Tworzy obiekt `std::string` (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.

```
constexpr std::string_view operator""sv(const char *str, std::size_t len) noexcept;
constexpr std::u16string_view operator""sv(const char16_t *str, std::size_t len)
noexcept;
constexpr std::u32string_view operator""sv(const char32_t *str, std::size_t len)
noexcept;
constexpr std::wstring_view operator""sv(const wchar_t *str, std::size_t len) noexcept;
```
Dostępny od wersji języka C++17. Tworzy obiekt `std::string_view` (lub inny, w zależności od rodzaju przekazywanego literału tekstowego) bezpośrednio odwzorowujący podany literał tekstowy jako obiekt mający cechy obiektowej zmiennej tekstowej (patrz opis klasy `std::string_view`).

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
   ....
}
```