

СИМВОЛЬНИЙ ТИП. РЯДКИ СИМВОЛІВ. ФУНКЦІЇ ОПРАЦЮВАННЯ ЛІТЕРНИХ РЯДКІВ

Символьний тип

Символами в C++ вважаються великі та малі літери, цифри, знаки арифметичних операцій, пробіл, розділові знаки та службові символи, які відповідають клавішам <Enter>, <Esc>, <Tab> тощо. *Типом символьних змінних* у C++ є тип **char**. При оголошенні змінних символьного типу в оперативній пам'яті для кожної з них виділяється по одному байту. Тобто для кодування символів використовується один байт (8 бітів), за допомогою якого можна закодувати $2^8 = 256$ різних символів.

Значення символьних констант записують в одиничних лапках.

Коди символів

Для уникнення розходжень при кодуванні символів розроблено єдиний міжнародний стандарт кодів для обміну інформацією – *таблиця ASCII-кодів* (American Standard Code for Information Interchange). Перші 128 ASCII-кодів (від 0 до 127) спільні для всіх країн, а наступні 128 кодів (від 128 до 255) утворюють розширену частину ASCII-таблиці, в якій залежно від країни розміщається національний алфавіт і символи псевдографіки. В ASCII-таблиці всі символи мають власний унікальний код і є впорядкованими.

Символ	Код
' ' (пробіл)	32
'0'...'9'	48...57
'A'...'Z'	65...90
'a'...'z'	97...122

Під час роботи з символами часто використовуються керуючі символи (<Esc>, <Enter>, <Tab> тощо), які не мають символьного подання, і називаються **недрукованими** (non-printed). Керуючі символи розташовані в перших 32-х кодах ASCII-таблиці. Звертатися до керуючих символів потрібно, використовуючи їхній код або ескейп- послідовність (escape). **Ескейп-послідовність** – це спеціальна комбінація символів, яка починається зі зворотної косої риски і записується в одиничних лапках.

Основні ескейп-послідовності

\n – переведення курсору на початок наступного рядка

\r – перехід на початок поточного рядка

\t – горизонтальна табуляція

`\v` – вертикальна табуляція

`\b` – видалення попереднього символу

`\\` – зворотна коса риска

`\"` – подвійні лапки

`\0` – завершальний символ рядка символів (нуль-символ)

Коли символні змінні отримують певні значення, то комп'ютер зберігає в пам'яті не самі символи, а їхні коди. Тому, якщо символній змінній присвоїти певне ціле число, то компілятор сприйме його як код певного символу з ASCII- таблиці. Це правило поширюється *лише на цілі числа*. Зважаючи на різні кодування розширеної частини ASCII-таблиці, для уникнення помилок при роботі з символами, потрібно використовувати їхнє символне подання, а не коди.

Наприклад, оголошення змінних

```
char c, d = 'Y', r = 115;
// виведення символу, що зберігається в змінній:
cout << "Output char variable d: " << d << endl; // Результат: Y
// виведення коду символу, що зберігається в змінній:
cout << "\nOutput code variable d: " << (int)d << endl; // Результат: 89
//виведення символу з кодом 115:
cout << "\n Output code variable r: " << r << endl; // Результат: s
// введення символу за допомогою cin:
cout << "\nEnter char: "; cin >> c;
int k = c;
//виведення коду символу, введеного з клавіатури:
cout << "Code of entered char: " << k;
```

На відміну від більшості мов програмування, в C++ дані типу `char` змінюються у діапазоні $-128 \dots 127$, причому числа $0 \dots 127$ є кодами символів спільної частини ASCII-таблиці, а від'ємні числа – розширеної частини ASCII- таблиці. Наприклад, літера кирилиці 'ю' має код -2 .

Крім типу `char`, для оголошення символних змінних (констант) також використовують тип `unsigned char` (беззнаковий символний тип), значення даних якого належать діапазону чисел $0 \dots 255$. В ASCII-таблиці значення кодів літер кирилиці більші за 127, тому символні змінні, значеннями яких є літери кирилиці, потрібно оголошувати як `unsigned char`.

При *порівнянні* символів більшим вважається той, код якого більший, тобто той символ, який у таблиці ASCII-кодів розміщений пізніше. Наприклад, справедливі такі нерівності: '5' < 'A', 'B' < 'C', 'C' < 'c', 'c' < 'z'.

Оскільки символний тип у C++ вважається цілим типом, то змінні цього типу можна *додавати та віднімати*. Результатом додавання двох символів буде символ, код якого дорівнює сумі кодів символів-доданків, а результатом різниці – символ, код якого дорівнює різниці кодів символів зменшуваного та від'ємника.

Функції для роботи з символами

Для опрацювання символьних даних використовуються стандартні бібліотеки C++ `<ctype.h>` та `<cctype>`, в яких визначені наступні функції.

Функція	Призначення
<code>tolower()</code>	повертає символ у нижньому регістрі
<code>toupper()</code>	повертає символ у верхньому регістрі
<code>isalnum()</code>	перевіряє чи належить символ множині латинських літер та цифр (A...Z, a...z, 0...9)
<code>isalpha()</code>	Перевіряє чи належить символ множині латинських літер (A...Z, a...z)
<code>isctrl()</code>	перевіряє чи належить символ множині керуючих символів (з кодами 0...31 та 127)
<code>isdigit()</code>	перевіряє чи належить символ множині цифр (0...9)
<code>isxdigit()</code>	перевіряє чи належить символ множині шістнадцяткових цифр (A...F, a...f, 0...9)
<code>isgraph()</code>	перевіряє чи належить символ множині друкованих символів, крім пробіла (<code>isalpha()</code> , <code>isdigit()</code> , <code>ispunct()</code>)
<code>islower()</code>	перевіряє чи належить символ множині латинських літер нижнього регістра (a...z)
<code>isupper()</code>	перевіряє чи належить символ множині латинських літер верхнього регістра (A...Z)

Так, наприклад, для зміни великої латинської літери *на малу* (переведення у нижній регістр) можна додати до значення символа різницю кодів між відповідними великими і малими літерами, яка дорівнює 32:

```
char c = c + 32;
```

або використати функцію, що зменшує регістр:

```
char c = tolower(c);
```

Розглянуті функції перевірки та зміни регістра працюють лише з латинськими літерами.

Рядки символів

На відміну від інших мов програмування, в мові C немає типу даних стрічка чи рядок. Тут рядок символів є послідовністю символів (масивом символів), який закінчується нуль-символом, тобто символом з кодом 0, що записується як керуюча послідовність `'\0'` і сприймається як один символ. Нуль-символ ще називають **нуль-термінальним** або **термінальним** (завершальним) символом.

Крім нуль-термінальних літерних рядків у стилі мови C, реалізованих на базі символьних масивів `char[]`, мова C++ дає змогу використовувати літерні рядки, реалізовані класом `string`.

Цей клас визначений у заголовному файлі `string`, тому для його використання необхідно вказати директиву підключення:

```
#include <string>
```

Розглянемо оголошення змінних

```
string s1("world"); // оголошує рядок s1 та ініціалізує його
string s2(10, 'b'); // оголошує рядок s2 та ініціалізує його
string s3(s1); // оголошує рядок s3 та ініціалізує його значенням s1
s1 += " hello"; // до рядка s1 додаємо C-рядок "hello"
string s4; // оголошує порожній рядок s4
char s[] = "word"; // оголошує C-рядок s та ініціалізує його л
s4 = 'D'; // рядку s4 присвоює символ 'D'
string s7(s + 6, s + 10); // оголошує рядок s7 та
```

Введення літерних рядків string

Як і для C-рядків, команда вводу літерного рядка `string`, яка виконується за допомогою безпосереднього використання потоку введення

```
string s;
cin >> s;
```

приведе до того, що будь-який пробільний символ (пробіл, табуляція, перехід до нового рядка) буде сприйматися як завершення вводу. Інакше кажучи, таким способом можна ввести лише одне (перше) слово деякого речення.

Для того, щоб мати можливість ввести пробіли та символи табуляції як значущі символи (тобто, щоб ввести ціле речення), необхідно використовувати функцію `getline()`, яка для рядків `string` має вигляд: `getline(cin, s);` – організовує ввід із консолі `cin` у літерний рядок `s`, для завершення вводу слід натиснути клавішу Enter (ввести новий рядок).

Операції над літерними рядками string

Літерні рядки `string` зручні у використанні. Для них визначені операції:

- = – присвоєння;
- + – конкатенація;
- += – конкатенація та присвоєння.
- == – порівняння на рівність;
- != – порівняння на нерівність;
- < – порівняння, чи менше;
- > – порівняння, чи більше;
- <= – порівняння, чи менше або дорівнює;
- >= – порівняння, чи більше або дорівнює

Операцію `+` можна використовувати для конкатенації одного літерного рядка `string` з іншим, або літерного рядка `string` із рядком в С-стилі.

Доступ до окремого символу рядка здійснюється за його індексом. Індексція починається з **нуля** (тобто, номер найпершого символу дорівнює нулю).

Функції для роботи з літерними рядками `string`

Функції опрацювання літерних рядків `string` зазвичай викликаються так:

```
string s;
s.<функція>(<аргументи>); або <змінна> = s.<функція>(<аргументи>);
```

Тобто, для виклику функції, яка має опрацювати літерний рядок `string`, зазвичай потрібно вказати: ім'я змінної, що містить літерний рядок; крапку; ім'я відповідної функції та її аргументи. Для літерних рядків `string` (наприклад, `s`) визначена константа `npos = -1` (або `s.npos`), яка представляє розмір рядка максимально можливої довжини. Якщо прототип функції закінчується ключовим словом `const`, то це означає, що ця функція не змінює літерного рядка.

`length()` та `size()` – повертають довжину (кількість символів) літерного рядка.

Функція `length()` збереглася від попередніх версій рядків `string`, а функція `size()` – відповідає сучасним бібліотекам C++.

`capacity()` – повертає поточну ємкість рядка – кількість символів, які рядок може містити до того, коли йому доведеться дати запит на виділення додаткової пам'яті.

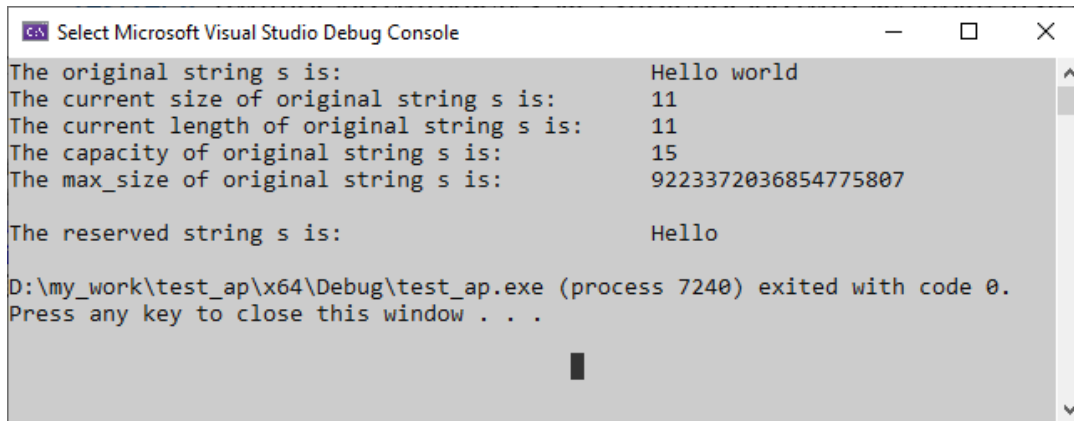
`max_size()` – повертає максимальну кількість символів, яку може містити рядок.

`resize()` – змінює розмір рядка: встановлює розмір, який визначається параметром `num`

Наприклад

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string s("Hello world");
    cout << "The original string s is: \t\t\t" << s << endl;
    // Порівнює функції size(), length(), capacity() та max_size()
    cout << "The current size of original string s is: \t"
         << s.size() << endl;
    cout << "The current length of original string s is: \t"
         << s.length() << endl;
    cout << "The capacity of original string s is: \t\t"
         << s.capacity() << endl;
    cout << "The max_size of original string s is: \t\t"
         << s.max_size() << endl << endl;
    s.erase(6, 5);
    // видалення частини рядка
    s.reserve(30);
    // після встановлення нового значення для ємкості рядка
    cout << "The reserved string s is: \t\t\t" << s << endl;
```

```
return 0;
}
```



```
Select Microsoft Visual Studio Debug Console

The original string s is:      Hello world
The current size of original string s is: 11
The current length of original string s is: 11
The capacity of original string s is: 15
The max_size of original string s is: 9223372036854775807

The reserved string s is:      Hello

D:\my_work\test_ap\x64\Debug\test_ap.exe (process 7240) exited with code 0.
Press any key to close this window . . .
```

Пошук

Є шість функцій пошуку, кожна з яких має чотири варіанти – які опрацьовують літерні рядки `string`, літерні рядки `char[]` (C-рядки `char*`), фрагменти літерних рядків `char[]` (C-рядків `char*`) та символи `char` відповідно.

`find()` – повертає позицію першого входження підрядка `str` у цьому рядку. Пошук починається з позиції `pos`. Якщо входження не виявлено, повертає значення `-1`

`rfind()` – повертає позицію останнього входження підрядка `str` у цьому рядку. Пошук починається з позиції `pos`. Якщо входження не виявлено, повертає значення `-1`.

`find_first_of()` – повертає позицію першого символу цього рядка, який збігається з будь-яким символом рядка `str`. Пошук починається з позиції `pos`. Якщо входження не виявлено, повертає значення `-1`

`find_last_of()` – повертає позицію останнього символу цього рядка, який збігається з будь-яким символом рядка `str`. Пошук починається з позиції `pos`. Якщо входження не виявлено, повертає значення `-1`.

`find_first_not_of` – повертає позицію першого символу цього рядка, який не збігається ні з яким символом рядка `str`. Пошук починається з позиції `pos`. Якщо входження не виявлено, повертає значення `-1`

`find_last_not_of()` – повертає позицію останнього символу цього рядка, який не збігається ні з яким символом рядка `str`. Пошук починається з позиції `pos`. Якщо входження не виявлено, повертає значення `-1`.

Порівняння

`compare()` – порівнює рядок `str` з цим рядком `s`. Повертає значення: менше нуля, якщо `s < str`; нуль, якщо `s == str`; більше нуля, якщо `s > str`.

Додавання

`append()` – додаває рядок `str` у кінець цього рядка. Повертає модифікований рядок.

Вставка

`insert()` – вставляє рядок `str` у позицію `pos` цього рядка. Повертає модифікований рядок.

Видалення

`erase()` – починаючи з позиції `pos`, видаляє `len` символів з цього рядка. Повертає модифікований рядок.

Заміна

`replace()` – замінює до `len` символів у цьому рядку, починаючи з позиції `pos`, рядком `str`. Повертає модифікований рядок

Копіювання

`copy()` – починаючи з позиції `pos`, копіює `len` символів із цього рядка в символьний масив, що адресується параметром `str`. Повертає кількість скопійованих символів.

Перевірка, чи рядок – порожній

`empty()` – повертає `true`, якщо цей рядок – порожній, та `false` у іншому випадку.