

# Символи і рядки

При виконуванні програм комп’ютер витрачає, за деякими оцінками, до 70 % часу на маніпулювання з текстовими рядками: копіює їх з одного місця пам’яті в інше, перевіряє наявність у рядках певних слів, поєднує чи відсікає рядки тощо. У C++ є два основні види рядків: С-рядки, які по суті є символьними масивами, і клас стандартної бібліотеки C++ `string`.

## 1 Символьний тип даних

Символами вважаються: великі й малі літери, цифри, знаки арифметичних дій ('+', '-', '\*', '/', '='), пробіл, розділові знаки ('.', ',', ';', ':', '!', '?', ' -'), службові символи, що відповідають клавішам <Enter>, <Esc>, <Tab> тощо. В C++ значення символічних констант записуються у одинарних лапках: '3', 'f', '+ ', '%'.

Для кодування усіх символів використовується восьмирозрядна послідовність 0 і 1, тобто один байт. Наприклад: символ цифри '9' кодується послідовністю бітів 0011 1001, символ літери латиниці 'w' - 0101 0111. За допомогою одного байта можна закодувати 256 різних комбінацій бітів, а отже, 256 різних символів.

Щоб не було розходжень у кодуванні символів, існує єдиний міжнародний стандарт - так звана таблиця ASCII-кодів (American Standard Code for Information Interchange - американський стандартний код для обміну інформацією, див. додаток А). Символи ASCII мають коди від 0 до 127, тобто значення першої половини можливих значень байта, хоча часто кодами ASCII називають всю таблицю з 256 символів. Перші 128 ASCII-кодів є єдині для всіх країн, а коди від 128 до 255 називають розширеною частиною таблиці ASCII, де залежно від країни розташовується національний алфавіт і символи псевдографіки.

У таблиці ASCII всі символи пронумеровано, тобто вони мають власний унікальний код. Так само як у кожній мові людського спілкування існує алфавіт (перелік усіх літер у чітко визначеному порядку), усі комп’ютерні символи теж є суверо упорядкованими. Символ ' ' (пробіл) має код 32, цифри мають коди від 48 для '0' до 57 для '9', великі латинські літери - від 65 для 'A' до 90 для 'Z', малі літери латиниці - від 97 для 'a' до 122 для 'z'. Кодування другої половини таблиці ASCII має різні варіанти. Найпоширенішими є DOS-кодування (866 кодова сторінка) і кодування 1251, яке є основним для Windows (див. додаток А). Звернімо увагу на різницю поміж цифрами і їхнім символічним зображенням. Наприклад, символ цифри '4' має ASCII-код 52 і не має безпосереднього відношення до числа 4.

Керувальні символи таблиці ASCII не мають символного подання, тобто не мають візуального зображення, тому їх інколи називають недрукованими (non-printed), наприклад: <Esc>, <Enter>, <Tab> тощо. Ці символи розташовано в перших 32-х кодах таблиці ASCII-кодів. Звертатися до таких символів можна через їхній код чи за допомогою так званої ескейп-послідовності (escape). Ес- кейп-послідовність - це спеціальна комбінація символів, яка розпочинається зі зворотної скісної риси і записується в одинарних лапках (табл. 7.1), наприклад: '\0', '\n'. Кожна з наведених у таблиці комбінацій символів вважається за один символ.

**Таблиця 1** Деякі поширені у застосуванні ескейп-послідовності

Символьне подання	Опис
\n	символ переведення курсора на початок наступного рядка
\r	переведення каретки
\t	символ переведення курсора на наступну позицію табуляції (відповідає клавіші <Tab>),
\b	символ вилучення попереднього символу перед курсором (відповідає клавіші <BackSpace>),
\a	символ звукового сигналу системного динаміка
\\\	символ \ (зворотна скісна риса)
\?	символ ? (знак запитання)
\'	символ ' (одинарні лапки)
\"	символ " (подвійні лапки)
\0	символ з кодом 0 є завершальним символом рядка символів
\_0xBісімкове число	символ, код якого зазначено у вісімковій системі числення
\_0xШістнадцяткове число	символ, код якого зазначено у шістнадцятковій системі числення

Тип символьних змінних у C++ називається **char**. Так, при оголошенні

```
char c, s, g;
```

в оперативній пам'яті для кожної з цих трьох змінних буде відведено по одному байту. Коли символьні змінні набувають певних значень, то комп'ютер зберігатиме в пам'яті не власне символи, а їхні коди. Наприклад, замість літери 'A' зберігатиметься її код 65. Тому, якщо присвоїти символьній змінній певне число, то C++ сприйме його як код символу з таблиці ASCII-кодів. Це поширюється лише на цілі числа. Зважаючи на різні кодування розширеної частини ASCII-таблиці для уникнення помилок вважається за оптимальне при роботі з символами використовувати їхнє символьне подання замість кодів. Наприклад, при оголошенні

```
char c = 115;
```

змінна c набуде значення символу 's', який має код 115.

Зауважимо, що у C++, на відміну від більшості мов програмування, дані типу **char** змінюються у діапазоні -128 ... 127, причому додатні числа 0 ... 127 зайняті символами спільної частини ASCII-таблиці, а символи розширеної частини ASCII-таблиці у C++ відповідають від'ємним числам. Наприклад, літера кирилиці 'ч' - має код -9, а кодом літери 'я' є -1.

Окрім типу **char**, існує його беззнакова модифікація **unsigned char**. Дані типу **unsigned char** мають значення у діапазоні 0 ... 255. В ASCII-таблиці значення кодів літер кирилиці є більшими за 127, тому, якщо треба мати справу зі змінними, значеннями яких є літери кирилиці, їх слід оголошувати типом **unsigned char**:

```
unsigned char c;
```

Символи можна порівнювати. Більшим вважається той символ, у якого код є більший, тобто символ, розташований у таблиці ASCII-кодів пізніше.

Наприклад: 'a' < 'h', 'A' < 'a'.

Оскільки символний тип `char` вважається у C++ за цілий тип, змінні цього типу можна додавати й віднімати. Результатом додавання буде символ, код якого дорівнює сумі кодів символів-доданків, наприклад:

```
char c = 'A';
char cl      = c +      5; // cl      = 'F'
char c2      = c +     32; // c2      = 'a'
char c3      = c -     10; // c2      = '7'
```

У C++ існує ціла низка спеціальних функцій для роботи з символними даними. Деякі з цих функцій наведено у табл. 7.2.

**Таблиця 2** Функції для роботи з символами

Функція	Призначення
<code>tolower()</code>	повертає символ у нижньому регістрі
<code>toupper()</code>	повертає символ у верхньому регістрі
<i>Належність символу до множини перевіряють такі функції:</i>	
<code>isalnum()</code>	латинських літер та цифр ('A' - 'Z', 'a' - 'z', '0' - '9')
<code>isalpha()</code>	латинських літер ('A' - 'Z', 'a' - 'z')
<code>iscntrl()</code>	керувальних символів (з кодами 0...31 та 127)
<code>isdigit()</code>	цифр ('0' - '9')
<code>isgraph()</code>	видимих символів, тобто не є відповідним до клавіш <Esc>, <Tab> тощо
<code>islower()</code>	латинських літер нижнього регістру ('a' - 'z')
<code>isprint()</code>	друкованих символів ( <code>isgraph()</code> + пробіл)
<code>isupper()</code>	літер верхнього регістру ('A' - 'Z')
<code>ispunct()</code>	знаків пунктуації
<code>isspace()</code>	символів-роздільників

Розглянемо деякі поширені алгоритми опрацювання символічних змінних.

1) Щоб визначити код символу, треба значення цього символу присвоїти цілій змінній. І, навпаки, щоб дізнатися, який символ відповідає певному числу, слід це число присвоїти символові. C++ сам виконає потрібні перетворення. Наприклад, після виконання команд `int x; char c = 'n';`  
`x = c;`

змінна `x` набуде значення 110, яке відповідає коду символу 'n' в ASCII-таблиці.

2) Визначити, чи є символ `c` цифрою, можна двома способами: перевірити його належність до символного проміжку від '0' до '9' за допомогою умови:

```
if(c>='0' && c<='9') . . .
```

або застосувати функцію `isdigit()` для перевірки належності символу до множини цифр:

```
if(isdigit(c)) . . .
```

3) Дізнатися, чи є символ `c` великою латинською літерою, можна теж двома способами: перевірити його належність до символного проміжку від 'A' до 'Z' за допомогою умови:

```
if(c>='A' && c<='Z') . . .
```

або застосувати функцію `isupper()` для перевірки належності символу до множини

великих латинських літер:

```
if(isupper (c)) . . .
```

4) Перевірку, чи є символ с **латинською ліteroю**, можна здійснити за допомогою умови

```
if(c>='A' && c<='Z' || c>='a' && c<='z') . . .
```

або за допомогою функції `isalpha ()`: `if(isalpha (c)) . . .`

5) Для перевірки, чи є символ с **малою латинською ліteroю**, слід записати умову

```
if(c>='a' && c<='z') . . .
```

або використати функцію `islower ()` для перевірки належності символу до множини малих латинських літер:

```
if(islower(c)) . . .
```

6) Для перетворення малої латинської літери с **на велику** (верхній регістр) можна використати функцію `toupper ()`:

```
c = toupper(c);
```

чи то відняти від значення літери різницю кодів між відповідними великими і малими літерами (вона становить 32):

```
char c = c - 32;
```

У аналогічний спосіб працює функція `tolower ()`, яка збільшує код великих літер латиниці на 32 й, отже, здобуває **нижній регістр** літер латиниці.

*Примітка.* Вищезазначені функції перевірки й перетворення регістру працюють лише з латинськими літерами. Для виконання дій 1 - 5 з літерами кирилиці слід використовувати перевірку належності символу до відповідного символьного проміжку, а для перетворення регістра - зменшення або збільшення коду символу на різницю кодів між великими і малими літерами (для більшості символів вона теж становить 32).

## 2 Рядки

Рядки у С являють собою послідовність (масив) символів із завершальним нуль-символом. Нуль-символ (нуль-термінатор) - це символ з кодом 0, який записується у вигляді керувальної послідовності '\0'. За розташуванням нуль-символу визначається фактична довжина рядка.

### 2.1 Масиви символів

Відмінною рисою символьного масиву є те, що в ньому насправді може бути менше символів, аніж зазначено при оголошенні. Окрім того, з цими масивами можна виконувати певні специфічні дії, які не можна здійснювати з числовими масивами (наприклад перевіряти наявність у масиві літери чи послідовності літер, копіювати масив як одне ціле, порівнювати масиви за алфавітом, дописувати один масив наприкінці іншого тощо).

Пам'ять під розміщення рядків, як і для будь-яких масивів, може виділятися як компілятором, так і динамічно - при виконуванні програми. Довжина динамічного рядка може задаватися змінною з визначенням заздалегідь значенням, а довжина статичного рядка має задаватися лише константою.

Рядок може бути оголошеним в один з нижче наведених способів:

- 1) `char *s; // Оголошення вказівника на перший символ рядка;`  
`// пам'ять під сам рядок не виділяється`
- 2) `char ss[15]; // Оголошення рядка ss з 15-ти символів;`

```

// пам'ять виділяється компілятором
3) const int n = 10;
char st[n]; // Оголошення рядка st з n (тобто 10-ти) символів;
// пам'ять виділяється компілятором
4) int n = 10;
char *str = new char[n]; // Оголошення рядка str з n (тобто 10)
// символів; пам'ять виділяється динамічно

```

При оголошенні рядок можна ініціалізувати рядковою константою, при цьому нуль-символ формується автоматично після останнього символу: `char str[10] = "Vasia";`

При цьому виділиться пам'ять під масив з 10-ти елементів та нуль-символ і перші 5 символів рядка записуються в перші 5 байт цієї пам'яті (`str[0] = 'V', str[1] = 'a', str[2] = 's', str[3] = 'i', str[4] = 'a'`), а в шостий елемент `str[5]` записується нуль-символ. Якщо рядок при оголошенні ініціалізується, його розмірність можна опускати (компілятор сам виділить потрібну кількість байтів):

```
char str[] = "Vasia"; // Виділено й заповнено 6 байтів
```

Рядки у лапках завжди неявно містять нуль-символ, тому при ініціалізації прописувати його немає потреби. Окрім того, різні наведені способи введення символьних масивів автоматично додають нуль-символ у кінець масиву.

При оголошенні й ініціалізації масиву слід бути впевненим, що розмір масиву є достатній, щоб умістити всі символи рядка з нуль-символом. Річ у тім, що функції, які опрацьовують рядки, керуються позицією нуль-символу, а не розміром рядка. C++ не накладає жодних обмежень на довжину рядка.

Звернімо увагу на те, що рядкова константа (у подвійних лапках) і символьна константа (в одинарних лапках) не є взаємозамінними. Це константи різних типів. Символ у одинарних лапках, наприклад, `'s'` є *символьною* константою. Для зберігання такої константи компілятор C++ виділяє лише один байт пам'яті. Символ у подвійних лапках, наприклад, `"s"` є *рядковою* константою, що окрім символу `'s'` містить символ `'\0'`, який додається компілятором. Більш того, `"s"` фактично являє собою адресу пам'яті, в якій зберігається рядок.

Як і числові масиви, символьні масиви опрацьовуються поелементно у циклі. Операція присвоювання одного рядка іншому є невизначена (оскільки рядок є масивом) і може виконуватися за допомогою циклу чи за допомогою функцій стандартної бібліотеки.

Для *введення* та *виведення* рядків у консолі використовуються `cin` і `cout`, наприклад:

```
const int n=10; char s[n];
cin>>s;
cout<<s;
```

Коли рядок виводиться за допомогою потоку `cout`, символи рядка виводяться по одному, доки не зустрінеться завершальний символ `'\0'`.

При введенні рядків у консолі замість оператора `>>` більш оптимально використовувати метод `getline ()`, оскільки потоковий оператор введення `>>` ігнорує пробіли. Окрім того, він може продовжувати введення елементів за межами масиву, якщо в пам'яті під рядок виділено менше місця, аніж вводиться символів.

Функція `getline()` має два параметри: перший аргумент - рядок, який вводиться, а другий - кількість символів.

Наприклад:

```
char s[4];
cout<<"Введіть рядок: "<<endl;
cin.getline(s, 4);
```

Аналогічні дії можна зробити використовуючи функцію `gets_s`:

```
char s[4];
cout<<"Введіть рядок: "<<endl;
gets_s(s);
```

Рядок `s` у цьому фрагменті програми може прийняти лише три значущих символи і буде завершений нуль-термінатором (символом '`\0`'). Решту введених символів рядка буде проігноровано.

Поширеним засобом доступу до символів рядка є вказівники типу `char*`. У прикладі

```
char *st = "Комп'ютерна програма";
компілятор записує всі символи рядка до масиву і присвоює змінній st адресу першого елемента масиву.
```

Рядок може вважатися за порожній у двох випадках: якщо вказівник на рядок має нульове значення `NULL` (немає взагалі жодного рядка) чи коли вказівник вказує на масив, який складається з одного нульового символу (не містить жодного значущого символу).

```
char *pc1 = 0;                                // pc1 не адресує жодного масиву символів,
const char *pc2 = "";                          // pc2 адресує нульовий символ
```

## Функції стандартної бібліотеки для роботи з рядками

C++ має багату колекцію функцій опрацювання рядків із завершальним нулем. Якщо в рядку відсутній нуль-термінатор, опрацювання рядка може тривати скільки завгодно, допоки в пам'яті не зустрінеться '`\0`'. У якості аргументів до функцій переважно передаються вказівники на рядки. Якщо при виконуванні функції здійснюється перенесення символів рядка з місця-джерела до місця-призначення, для рядка-призначення слід завчасно зарезервувати місце в пам'яті. Копіювання рядків з використанням просто вказівника, а не адреси початку завчасно оголошеного масиву - одна з найпоширеніших помилок програмування, навіть у досвідчених програмістів. При виділенні місця для рядка- призначення слід виділяти місце і для нуль-термінатора.

У табл. 7.3 наведено функції стандартної бібліотеки для роботи з С-рядками. Деякі з цих функцій, наприклад `strlen()`, `strcpy()`, опрацьовують рядки, оголошені в будь-який з чотирьох раніш розглянутих способів. Але більшість функцій потребує, щоб рядок був оголошений як вказівник типу `char*` (див. способи 1 і 4 на початку).

У консолі для використання цих функцій слід залучити до програми бібліотеку `<cstring>`. Перелік основних функцій цієї та інших стандартних бібліотек C++ наведено в таблиці

**Таблиця 3** Функції стандартної бібліотеки `<cstring>`

Функція	Призначення	Формат
<code>strlen()</code>	повертає довжину рядка (без урахування символу завершення рядка)	<code>size_t strlen (char *s) ;</code>
<code>strcat()</code>	долучає <code>s2</code> до <code>s1</code> і, як результат, повертає <code>s1</code>	<code>char *strcat(char *s1, char *s2);</code>
<code>strncat()</code>	долучає до рядка <code>s1</code> перші <code>n</code> символів з рядка <code>s2</code>	<code>char * strncat (char *s1, char *s2, size_t n);</code>
<code>strlwr()</code>	перетворює всі латинські літери до нижнього регістру	<code>char *strlwr(char *s);</code>
<code>strupr()</code>	перетворює всі латинські літери до верхнього регістру	<code>char *strupr(char *s);</code>
<code>strcmp()</code>	порівнює рядки і повертає нульове значення, якщо рядки є однакові ( <code>s1=s2</code> ), від'ємне ( <code>s1&lt;s2</code> ) чи додатне ( <code>s1&gt;s2</code> ). Порівняння відбувається посимвольно і в якості результату повертається різниця кодів перших неоднакових символів	<code>int *strcmp(char *si, char *s2);</code>
<code>strncmp()</code>	на відміну від попередньої функції, порівнює лише перші <code>n</code> символів рядка <code>si</code> з <code>n</code> символами рядка <code>s2</code>	<code>int *strncmp (char *si, char *s2, size_t n);</code>
<code>stricmp()</code>	порівнює два рядки, не розрізнюючи прописні й малі літери латиниці	<code>int stricmp (char*si, char *s2);</code>
<code>strnicmp()</code>	порівнює лише перші <code>n</code> символів двох рядків, не розрізняючи великі й малі літери латиниці	<code>int strnicmp (char *si, char *s2, size_t maxlen);</code>
<code>strcpy()</code>	копіює до <code>si</code> рядок <code>s2</code> , повертає <code>si</code> , при цьому попереднє значення <code>si</code> втрачається	<code>char *strcpy (char *si, char *s2);</code>
<code>strncpy()</code>	замінює перші <code>n</code> символів рядка <code>s1</code> на перші <code>n</code> символів рядка <code>s2</code>	<code>char *strncpy (char *si, char *s2, size_t n);</code>
<code>strchr()</code>	відшукує перше входження символу <code>ch</code> в рядок <code>s</code> і повертає вказівник на цей символ, тобто частину рядка <code>s</code> , розпочинаючи з символу <code>ch</code> і до кінця рядка. Якщо символу <code>ch</code> в рядку <code>s</code> немає, результат - <code>NULL</code>	<code>char *strchr(char *s, int ch);</code>

strrchr()	відшукує останнє входження символу в рядку і повертає частину рядка s, розпочинаючи з останнього входження символу ch і до кінця рядка. Якщо символу ch в рядку s немає, результат - NULL	char * <b>strrchr</b> (char *s, int c);
strstr()	відшукує підрядок s2 в рядку si, повертає частину рядка si, розпочинаючи з першого спільногом символу для обох рядків і до кінця si	char * <b>strstr</b> (char *si, char *s2);
strspn()	повертає довжину початкового сегмента рядка si, символи якого є в рядку s2	size_t <b>strspn</b> (char *si, char *s2);
strcspn()	повертає індекс першого символу в рядку si, який є спільним для обох рядків (нумерація індексів символів розпочинається з нуля)	size_t <b>strcspn</b> (char *si, char *s2);
strset()	замінює усі символи рядка s на символ c	char * <b>strset</b> (char *s, char c);
strnset()	замінює лише перші n символів рядка s на символ c	char * <b>strnset</b> (char *s, int ch, size_t n);
strupbrk()	відшукує місце першого входження у рядок s1 будь-якого символу рядка s2 і повертає частину рядка s1, розпочинаючи з цього символу і до кінця рядка	char * <b>strupbrk</b> (char *s1, const char *s2);
strrev()	реверс рядка s	char * <b>strrev</b> (char *s);
strtok()	повертає частину (лексему) рядка s1 від поточної позиції вказівника до розділового символу, зазначеного у рядку s2; зазвичай використовується для перетворювання рядка на послідовність лексем	char * <b>strtok</b> (char *s1, const char *s2);

Розглянемо деякі з наведених у табл. 3 функцій детальніше на прикладах, для чого попередньо оголосимо вказівники на рядки s1, s2 та s3 і надамо їм початкові значення.

```
int k, n; char      *s1 = "На Дерибасівській", *s2="гарна погода", *s3;
      n = strlen(s1);      // Обчислюється довжина рядка si; n дорівнюватиме 17
      k = strlen(s2);      // Обчислюється довжина рядка s2; k=13
      k = strcmp(s1, s2);   // k=173; різниця між ASCII-кодами перших неоднакових
                           // символів рядків s1 і s2 дорівнює 173, а оскільки 173>0,
                           // можна стверджувати, що s1>s2
      strcpy(s3, s2);       // s3=" гарна погода" (рядку s3 присвоюється рядок s2)
      strncpy(s3, s1, 2);   // Копіюються два перші символи з рядка s1 до s3
      s3[2] = '\0';         // Третім символом після символів "На" задається
                           // нуль-символ щоби обрізати рядок
      strcat(s1, s2);       // s1 = "На Дерибасівській гарна погода"; до рядка s1
                           // долучається рядок s2
```

```

s3 = strchr(s1, ' '); // Пошук пробілів у рядку s1; тепер s3=" Дерибасівській
                      // "гарна погода" - це рядок від першого пробілу до кінця
                      // рядка s1, тобто без першого слова
s3=strrchr (s1, ' '); // Пошук останнього пробілу в рядку s1;
                      // тепер s3=" погода" - це останнє слово рядка s1
n=strchr (s1, ' ') - s1+1; // n=3 - індекс першого пробілу в рядку s1, обчислений
                           // як різниця вказівників
k = strcspn(s1, " ") + 1; // k=3 - індекс першого пробілу в рядку s1; оскільки
                           // нумерація індексів символів розпочинається з 0, слід додати 1
s3=strstr(s1,s2); // s3 = "гарна погода"; пошук рядка s2 у рядку s1
s3=strupr("адогон анраг"); // s3="адогон анраг" - відбувається реверс рядка
s3 = strupr("C++ ms"); // s3="C+ + MS " - перетворення усіх
                           // латинських літер рядка до верхнього регістру
s3 = strlwr(s3); // s3="c+ + ms" – зворотне перетворення усіх
                           // латинських літер рядка до нижнього регістру
s3 = strtok(s1, " "); // s3="На" - перше слово (лексема) рядка s1 до розділового
                           // знака, який вказано символом (пробілом) у другому
                           // аргументі функції. Почергове здобуття усіх лексем рядка можна організовувати у циклі

```

Отже, функції `strcpy()` та `strncpy()` призначено для копіювання рядка чи то його частини до іншого рядка. Функції `strchr()`, `strrchr()` та `strstr()` повертають вказівник на віднайдений символ чи підрядок.

Функція `strtok()` використовується для перетворювання рядка на послідовність лексем. *Лексема* - це послідовність символів, відокремлених символами-розділювачами (зазвичай пробілами чи знаками пунктуації). Наприклад, у рядку тексту кожне слово може розглядатися як лексема, а пробіли, що відокремлюють слова одне від одного, можна розглядати як розділювачі. Для того щоб розбити рядки на лексеми, треба організувати кілька викликів функції `strtok()` (за умови, що рядок вміщує понад одну лексему).

## The ASCII code

American Standard Code for Information Interchange