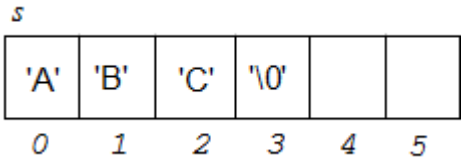
Лекція 23

Тип даних «текстовий рядок»

Рядок – послідовність (масив) символів. Якщо у виразі зустрічається одиночний символ, він повинен бути укладений в одинарні лапки. При використанні у виразах рядок береться в подвійні лапки. Ознакою кінця рядка є нульовий символ \0. У С ++ рядки можна описати за допомогою масиву символів (масив елементів типу char), в якому слід передбачити місце для зберігання ознаки кінця рядка.

Нехай заданий рядок ***s*** (див. мал.):



Пам'яті під неї відведемо 6 байт, тому опис буде наступним:

```
char s[6];
```

Корисна інформація (текст "ABC") розміщується в перших трьох байтах, а потім в третьому байті знаходиться нуль-символ '\ 0'.

Рядок ***s*** можна було б ініціювати, тобто на етапі компіляції задати їй початкове значення:

```
char s[6] = {'A', 'B', 'C', '\0'};
```

Так як ***s*** – це масив (хоча і символів), тому ініціалізація записана точно так же, як це зазвичай робиться для масиву. Але ***s*** – це ще й рядок, тому допускається спрощений варіант ініціалізації, схожий з ініціалізацією простих змінних:

```
char s[6] = "ABC";
```

При ініціалізації для рядка допустимо не задавати обсяг виділеної пам'яті:

```
char s[] = "ABC";
```

Тепер компілятор сам підрахує, скільки потрібно відвести пам'яті під рядок, і виділить під неї необхідний обсяг.

Коли пам'ять під рядок виділена, з цим рядком можна починати працювати. Рядки – вельми своєрідні об'єкти. З одного боку – це масиви, тому можна звертатися за індексом до потрібної комірки, змінювати її вміст, наприклад:

```
s[1]='i';
```

В результаті рядок ***"ABC"*** перетвориться в ***"AiC"***.

З іншого боку, робота з рядком схожа на роботу з простою змінною, наприклад, виведення рядка на екран монітора (використовуємо стандартну операцію виведення мови С ++):

```
cout << "s=" << s << endl;
```

Погодьтеся, що це набагато простіше, ніж виведення з використанням циклу:

```
for(i = 0; i < n; i++)
   cout << x[i] << endl;
```

Дії з рядками

1. Виведення рядка на екран монітора

1 а) Використовуємо стандартну операцію виведення мови С ++:

```
cout << "s=" << s << endl;
```

б) Використовуємо функцію виведення на екран символьного рядка (функція мови С):

```
puts(s);
```

2. Введення рядка

2а) Використовуємо стандартну операцію введення мови С ++:

```
cin >> s;
```

Таким способом можна прочитати зі стандартного пристрою тільки рядок без пропусків, так як при використанні операції введення (>>) саме пропуск є роздільником, що дозволяє послідовно зчитувати дані в дві або більше змінні, наприклад:

```
int a,b;
cin >> a >> b;
```

2б) Введення рядка, що містить пропуски:

```
gets(s);
```

Читається будь-яка інформація до натискання клавіші "Enter". Символи кінця введення (їх коди – це десяткові числа 13 (повернення каретки) і 10 (новий рядок)) замінюються на нуль-символ, але тут немає перевірки на кількість введеної інформації. Якщо пам'яті під рядок ***s*** відведено 6 байт (як на малюнку вище), то введення більш, ніж 5 символів створить проблему. Мови С / С ++ не контролюють вихід за межі масиву, тому введена інформація може потрапити на сусідні ділянки пам'яті.

3. Обчислення довжини рядка:

```
int k;
k = strlen(s);
```

тут ***k*** – це поточна довжина рядка, тобто кількість корисної інформації (формально – кількість елементів до нуль-символу). Наприклад, для рядка

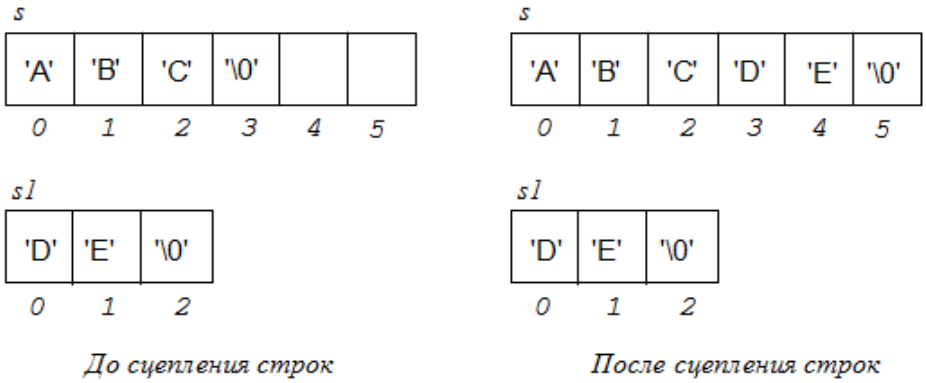
```
char s[6] = "ABC";
```

k дорівнюватиме 3.

4***. Об'єм пам'яті, відведеної під рядок***, можна обчислити за формулою:

```
int size = sizeof(s)/sizeof(char);
```

5. ***Зчеплення рядків*** (конкатенація)




```
strcat(s, s1);
```

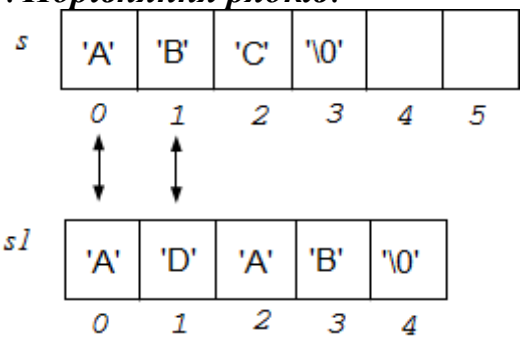
Тут в кінець рядка ***s*** додається вміст рядка ***s1***. Результат зберігається в рядку ***s***. Рядок ***s1*** не змінюється.

6. ***Копіювання рядків***:

```
strcpy(s, s1);
```

Дані з рядка ***s1*** побайтно копіюються в рядок ***s***. Колишній зміст рядка ***s*** втрачається, а рядок ***s1*** залишається незмінною.

7. ***Порівняння рядків***:



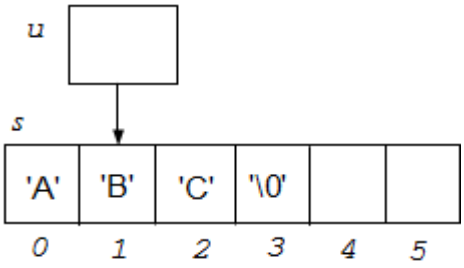

```
char s[6] = "ABC";
char s1[] = "ADAB";
int k = strcmp(s, s1);
```

У функції **strcmp()** виконується побайтне порівняння кодів символів: ***s***[0] і ***s1***[0] рівні – йдемо далі; ***s***[1] і ***s1***[1] – не рівні, код символу 'D' більше коду 'B', отже рядок ***s1*** більше ***s***.

Результат роботи функції:

- k = 0 - рядки рівні;
- k <0 - рядок s менше ***s1***;
- k> 0 - рядок s більше ***s1***.

8. ***Пошук першого входження символу*** в рядку:

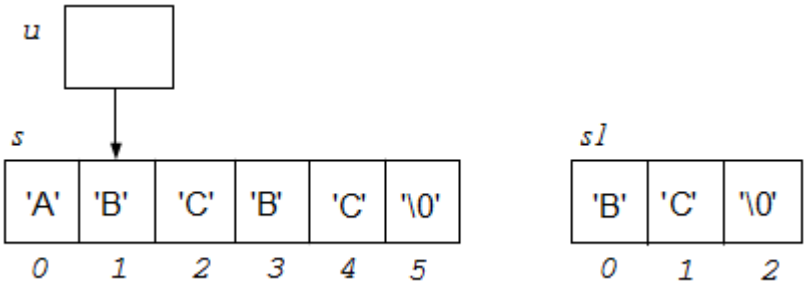



```
char *u;
        char c = 'B';
        u = strchr(s, c);
```

Результатами виконання функції **strchr()** буде адреса першого символу в рядку ***s***, який дорівнює шуканого символу ***c***. Якщо символ не знайдено, то в покажчик ***u*** буде записано число 0.

9. ***Пошук першого входження підрядка*** в заданому рядку:

```
u = strstr(s, s1);
```



Тут в рядку ***s*** підрядок ***"BC"*** зустрічається двічі, але в покажчик ***u*** буде записана адреса першої з них (з індексу, рівного 1).

Зауваження по роботі з рядками

1) Якщо довжина рядка змінюється стандартними функціями типу **strcat()** або **strcpy()**, а засобами користувачами, то необхідно самому додавати в кінець рядка нуль-символ.

Приклад:

```
        char s[6];
int i;
for(i = 0; i < 4; i++)
   s[i] = i + 'A';
s[i]='\0';
```

Заповнюємо ***s*** символьним рядком ***"ABCD"***. Після виходу з циклу змінна ***i*** дорівнює 4. У цю позицію і записуємо нуль-символ.

2) Якщо в пам'яті вихідні рядки перетинаються, то для ряду функцій, таких, як копіювання, зчеплення, можливі побічні ефекти, і результат роботи функцій не передбачуваний.

3) У цій темі наведені приклади використання тільки деяких функцій по роботі з рядками. Існує цілий ряд функцій, що дозволяють обробити тільки частину рядка, наприклад:

```
int n1=2;
strncat(s, s1, n1);
```

Додати в кінець рядка ***s*** перші ***n1*** символів з рядка ***s1***. У таких функціях (**strncat()**, **strncmp()** та інші) в середині назви записаний символ ***n***.

Зазвичай обробка ведеться з початку рядка, але є варіанти функцій для обробки рядків з кінця, наприклад:

```
char *u;
char c = 'B';
u = strrchr(s, c);
```

Обчислюється адреса останнього входження символу ***c*** в рядок ***s***. Тут в середині назви функції (**strrchr()**) присутній символ ***r*** (скорочення слова **righ**t – правий).

4) Для обробки рядків необхідно підключати заголовки

```
#include <string.h>
```

який забезпечує доступ до прототипів всіх необхідних функцій по роботі з рядками.

5) Якщо застосовуються функції читання-запису символів (**getchar()**, **putchar()**) або рядків (**gets()**, **fgets(), puts()**), то необхідно підключати заголовки

```
#include <stdio.h>
```

Приклади завдань з рядками

Працюючи з рядками, необхідно дотримуватися простих правил:

– всюди, де є така можливість, використовуємо стандартні функції по обробці рядків;

- якщо немає підходящої стандартної функції – працюємо з рядком як з масивом;

- не забуваємо про нуль-символ в кінці рядка.

Приклад 1. Дана символьний рядок, що не містить пропусків. Замінити всі символи '+', розташовані за першою літерою 'A', на символи '*'.

```
#include <iostream>
using namespace std;
#include <string.h>
#include <stdio.h>
int main()
{  char s[30];
   cout << "Input s:" << endl;
   cin >> s;
   char *u = strchr(s,'A');
   if(u)
   {  u++;
      for(int i=0; i < strlen(u); i++)
         if(u[i] == '+')
            u[i] = '*'; }
   cout <<"s=" << s << endl;
```

```
   return 0;}
```

Приклад 2. Дано символьний рядок. Підрахувати кількість слів, з яких складається цей рядок. Під поняттям «слово» будемо мати на увазі послідовність будь-яких символів, що вводяться з клавіатури (або з текстового файлу), розділених одним або декількома пропусками.

```
#include <iostream>
using namespace std;
#include <string.h>
#include <stdio.h>
int main()
{ char s[80];
   cout << "Input s:" << endl; gets(s);
   int i, d = strlen(s), k = 0;

   if(d > 0)
   {  for(i = 0; i < d - 1; i++)
       if(s[i] != ' ' && s[i+1] == ' ') k++;
      if(s[d-1] != ' ') k++;
   }
   cout <<"k=" << k << endl;
   return 0;}
```

Рядки Visual Studio

Для роботи з рядками у середовищі **Visual Studio** передбачений клас **string** призначений для роботи з рядками типу **char***, які представляють собою рядок із завершальним нулем.

Щоб скористатися наявними можливостями класу **string** в **MS Visual Studio (C ++)**, потрібно підключити бібліотеку **<string>** і простір імен **std**.

```
#include <string>
using namespace std;
```

Як і будь-який клас, клас string має ряд конструкторів. Основні з них такі:

```
string();
string(const char * str);
string(const string & str);
```

Нижче наведені приклади ініціалізації змінних типу **string**

```
string s1("Hello!"); // инициализация - конструктор string(const char * str)
string s2 = "Hello!"; // инициализация - конструктор string(const char * str)
char * ps = "Hello";
string s3(ps); // инициализация
string s4(s3); // инициализация - конструктор string(const string & str)
string s5; // инициализация - конструктор string()
```

Щоб присвоїти один рядок іншому, можна застосувати один з двох методів:

- використовувати оператор присвоювання '=';

- використовувати функцію **assign()** з класу **string**.

Функція **assign()** має кілька реалізацій.

- Перший варіант – це виклик функції без параметрів:

string &assign(void);

В цьому випадку відбувається просте присвоювання одного рядка інший.

- Другий варіант дозволяє копіювати задану кількість символів з рядка:

string &assign(const string & s, size_type st, size_type num);

де

- ***s*** – об'єкт, з якого береться вихідна рядок;

- ***st*** – індекс (позиція) в рядку, з якої починається копіювання num символів;

num – кількість символів, які потрібно скопіювати з позиції st;

- ***size_type*** – порядковий тип даних.

- Третій варіант копіює в об'єкт перші ***num*** символів рядка

string & assign(const char * s, size_type num);

де

- ***s*** – рядок, яка завершується символом '\ 0';

- ***num*** – кількість символів, які копіюються в об'єкт з рядка s.

Приклад.

```
// присваивание строк, функция assign()
string s1 = "bestprog.net";
string s2;
string s3;
char * ps = "bestprog.net";

s3 = s1; // s3 = "bestprog.net"
s2.assign(s1); // s2 = "bestprog.net"
s2.assign(s1, 0, 4); // s2 = "best"
s2.assign(ps, 8); // s2 = "bestprog"
```

Об’єднання рядків

Крім розглянутої функції конкатенації для об'єднання рядків використовується функція **append()**. Для додавання рядків також можна використовувати операцію '+', наприклад:

```
string s1;
string s2;
s1 = "abc";
s2 = "def";
s1 = s1 + s2; // s1 = "abcdef"
```

Функція **append()** добре підходить, якщо потрібно додавати частину рядка. Функція має такі варіанти реалізації:

```
string &append(const string & s, size_type start);
string &append(const char * s, size_type num);
```

Приклад.

```
string s1 = "abcdef";
s2 = "1234567890";
```

```
append(s2, 3, 4); // s1 = "abcdef4567"

char * ps = "1234567890";
s1 = "abcdef";
s1.append(ps, 3); // s1 = "abcdef123"
```

Вставка символу у рядок

Щоб вставити один рядок в задану позицію іншого рядка потрібно використовувати функцію insert (), яка має кілька варіантів реалізації.

Перший варіант функції дозволяє вставити повністю весь рядок **s** в задану позицію ***start*** викликає рядки (що викликає об'єкта):

```
string & insert(size_type start, const string &s);
```

Другий варіант функції дозволяє вставити частину (параметри ***insStart***, ***num***) рядка ***s*** в задану позицію ***start*** :

```
string & insert(size_type start, const string &s, size_type insStart, size_type num);
```

де

- ***s*** – рядок, яка вставляється у визиваючий рядок;
- ***start*** – позиція викликаючого рядка, з якої здійснюється вставка рядка ***s***;
- ***insStart*** – позиція в рядку ***s***, з якої відбувається вставка;
- ***num*** – кількість символів в рядку ***s***, які вставляються з позиції ***insStart***.

Приклад.

```
string s1 = "abcdef";
string s2 = "1234567890";

s1.insert(3, s2); // s1 = "abc"+"1234567890"+"def"="abc1234567890def"
s2.insert(2, s1, 1, 3); // s2 = "12bcd34567890"
```

Заміна символів у рядку

Функція ***replace()*** виконує заміну символів в викликає рядку. Функція має такі варіанти реалізації:

```
string &replace(size_type start, size_type num, const string &s);
string &replace(size_type start, size_type num, const string &s, size_type replStart,
size_type replNum);
```

У першому варіанті реалізації рядок, що викликає замінюється рядком ***s***. Є можливість задати позицію (***start***) і кількість символів (***num***) в викликаючому рядку, які потрібно замінити рядком ***s***.

Другий варіант функції **replace()** відрізняється від першого тим, що дозволяє замінювати викликаючий рядок тільки частиною рядка ***s***. В цьому випадку задаються два додаткові параметри: позиція ***replStart*** і кількість символів в рядку ***s***, які утворюють подстроку, яка замінює викликає рядок.

```
string s1 = "abcdef";
string s2 = "1234567890";

s2.replace(2, 4, s1); // s2 = "12abcdef7890"
s2 = "1234567890";
s2.replace(3, 2, s1); // s2 = "123abcdef67890"
s2 = "1234567890";
```

```
s2.replace(5, 1, s1); // s2 = "12345abcdef7890"
```

Видалення заданої кількості символів з рядка.

Для видалення символів з викликає рядки використовується функція **erase():**

```
string & erase(size_type index=0, size_type num = npos);
```

де

- ***index*** – індекс (позиція), починаючи з якої потрібно видалити символи в викликаючому рядку;

- ***num*** – кількість символів, які видаляються.

Приклад.

```
string s = "01234567890";
s.erase(3, 5); // s = "012890"
s = "01234567890";
s.erase(); // s = ""
```

Пошук символу в рядку.

У класі **string** пошук рядка в підрядку можна робити двома способами, які відрізняються напрямком пошуку:

- шляхом перегляду рядка від початку до кінця за допомогою функції **find()**;

- шляхом перегляду рядка від кінця до початку функцією **rfind().**

Прототип функції **find()** має вигляд:

```
size_type find(const string &s, size_type start = 0) const;
```

де

- ***s*** – підрядок, який шукається в рядку, що викликає цю функцію. Функція здійснює пошук першого входження рядка ***s***. Якщо підрядок ***s*** знайдений в рядку, тоді повертається позиція першого входження. В іншому випадку повертається -1;

- ***start*** – позиція, з якої здійснюється пошук.

Прототип функції **rfind()** має вигляд:

```
size_type rfind(const string &s, size_type start = npos) const;
```

де

- ***s*** – підрядок, який шукається. Пошук підрядка в рядку здійснюється від кінця до початку. Якщо підрядок ***s*** знайдено, то функція повертає позицію першого входження. В іншому випадку функція повертає -1;

- ***npos*** – позиція останнього символу викликаючого рядка;

- ***start*** – позиція, з якої здійснюється пошук.

Приклад.

```
1. // тип string, функция find()
string s1 = "01234567890";
string s2 = "345";
string s3 = "abcd";
int pos;

pos = s1.find(s2); // pos = 3
```

```
pos = s1.find(s2, 1); // pos = 3
pos = s1.find("jklmn", 0); // pos = -1
pos = s1.find(s3); // pos = -1
pos = s2.find(s1); // pos = -1
```

2.
```
// тип string, функции find() и rfind()
string s1 = "01234567890";
string s2 = "345";
string s3 = "abcd";
string s4 = "abcd---abcd";
int pos;

pos = s1.rfind(s2); // pos = 3
pos = s1.rfind(s2, 12); // pos = 3
pos = s1.rfind(s2, 3); // pos = 3
pos = s1.rfind(s2, 2); // pos = -1
pos = s2.rfind(s1); // pos = -1
pos = s1.rfind(s3, 0); // pos = -1

// разница между функциями find() и rfind()
pos = s4.rfind(s3); // pos = 7
pos = s4.find(s3); // pos = 0
```

Порівняння рядків

Оскільки тип **string** є класом, то, щоб порівняти два рядки між собою можна використовувати операцію '= ='. Якщо два рядки однакові, то результат порівняння буде ***true***. В іншому випадку, результат порівняння буде ***false***.

Але якщо потрібно порівняти частину одного рядка з іншим, то для цього передбачена функція **compare ()**.

Прототип функції **compare()**:

```
int compare(size_type start, size_type num, const string &s) const;
```

де

- ***s*** – рядок, яка порівнюється із зухвалою рядком;
- ***start*** – позиція (індекс) в рядку ***s***, з якої починається перегляд символів рядка для порівняння;
- ***num*** – кількість символів в рядку ***s***, які порівнюються із викликаючим рядком.

Функція працює наступним чином. Якщо викликаючий рядок менше рядка ***s***, то функція повертає -1 (від'ємне значення). Якщо викликаючий рядок більше рядка ***s***, функція повертає 1 (позитивне значення). Якщо два рядки рівні, функція повертає 0.

Приклад.

```
// тип string, функция compare()
string s1 = "012345";
string s2 = "0123456789";
int res;

res = s1.compare(s2); // res = -1
```

```
res = s1.compare("33333"); // res = -1
res = s1.compare("012345"); // res = 0
res = s1.compare("345"); // res = -1
res = s1.compare(0, 5, s2); // res = -1
res = s2.compare(0, 5, s1); // res = -1
res = s1.compare(0, 5, "012345"); // res = -1
res = s2.compare(s1); // res = 1
res = s2.compare("456"); // res = -1
res = s2.compare("000000"); // res = 1
```