**ЛАБОРАТОРНА РОБОТА № 5.**

**Рядки в мові програмування С++**

# 5.1. Мета роботи

Навчитися використовувати символьні дані для розв’язання задач роботи з рядками.

# 5.2. Теоретичні відомості

C++ надає два типи представлень рядків:

* рядок символів у стилі C.
* тип класу String, представлений стандартним C++.

5.2.1. Рядки символів у стилі C

У C не визначено спеціального типу для опрацювання рядків. Масив символів (чи рядок або стрічка) розглядається як масив елементів типу **char**, який закінчується символом **'\0'** (нуль-символ) що є ознакою кінця рядка. Такі рядки називають ASCII-рядками*.* Сталі типу рядок записують у лапках, наприклад, **"Lviv Polytechnic"**, **"students"**, **" "** – рядок, що містить один символ-пробіл. У таких сталих рядках нуль-символ дописується автоматично.

Масиви символів оголошують так:

**char <назва рядка>[довжина рядка];**

Підчас оголошення символьного масиву необхідно до фактичної довжини рядка додати одиницю для нульового символу. Якщо масив символів оголошують й ініціалізують одночасно, то довжину можна не зазначати, компілятор визначить її сам. Оскільки рядки є масивами символів, то назва рядка є вказівником-константою на його перший елемент (на перший символ). Розглянемо оголошення та ініціалізацію рядків:

|  |
| --- |
| **const char text1[] = "We are learning C language"; char text2[] = "We" " are learn"**  **"ing C language ";**  **char word[] = "University"; char lexem1[11], lexem2[40];** |

Тут оголошено константу **text1**, яка має значення **"We are learning C language"**, символьні масиви: **word** (без зазначення розміру), **lexem1** (може містити до 10 символів) та **lexem2** (до 39 символів).

Символьний масив **word** ще можна оголосити так:

**char word[11] = "University";**

або так:

**char word[ ] = {**

**'U','n','i','v','e','r','s','i','t','y','\0'};**

Тут потрібно *вручну* записати нуль-символ, інакше змінна **word** трактуватиметься не як рядок, а як звичайний масив, і при спробі його вивести, на екран, зазвичай буде виведено все "сміття", що знаходиться в пам’яті після адрес відведених під рядок, аж поки не зустрінеться перший нуль-символ. Наприклад:

|  |
| --- |
| **double pi = 3.14159;**  **char word[10] = {**  **'U','n','i','v','e','r','s','i','t','y'}; printf("%s", word);** |

Виведе щось подібне до:

**UniversitynЖ←Ё∙! @**

Це відбулося тому, що компілятор розмістив в наступних комірках число **pi**, а функція **printf()** спробувала вивести його на екран побайтно у вигляді символів.

Рядки можна опрацьовувати посимвольно за допомогою назви масиву, наприклад, так:

|  |  |
| --- | --- |
| **for (int n = 0; n < 11; n++)** | |
|  | **\*(lexem1+n) = \*(word+n);** |
| **printf("%s", lexem1);** | |

Змінній **lexem1** надається значення **"University"** і ця фраза виводиться на екран. Інакше це можна зробити так:

**for (int n = 0; n < 11; n++) lexem1[n] = word[n];**

**printf("%s", lexem1);**

Ввести весь масив символів можна за допомогою команди (**'\0'** додається автоматично):

**scanf("%s", <назва масиву>);**

Якщо рядок даних містить символ пропуску (**' '**, **'\t'**, **'\n'**, **'\v'**, **'\f'**, **'\r'**), то **scanf()** зчитає дані лише до першого пропуску. Щоб ввести весь рядок до символу вводу, можна або задати спеціальне форматування з явним вказанням символів, що необхідно ігнорувати, або застосувати функції бібліотеки **<cstdio>**:

**char \* gets(char \* str);**

**int puts(const char \*str);**

При вдалому виконанні функція **gets()** повертає рядок, при невдалому нуль. Також нуль буде повернуто, якщо у рядку зустрінеться **EOF.**

Обидві функції не перевіряють розмір буферу, куди буде записаний рядок, тому відповідальність за можливе переповнення покладається на програміста. Більше того, стандарт C11 вважає функції типу **gets()** та **puts()** застарілими та строго не рекомендує їх використовувати. Для уникнення таких випадків можна використовувати функції для роботи з файлами **fgets()** та **fputs()**:

**char \* fgets( char \* str, int num, FILE \* stream);**

**int fputs(const char \* str, FILE \* stream );**

**fputs()** повертає істине значення при вдалому виклику і **EOF** при невдалому.

Слід зауважити, що ці функції орієнтовані на роботу з файлами, тому мають ряд особливостей:

* **fgets()** приймає другим аргументом максимальну кількість символів для читання. Читання відбувається до першого символу переходу на новий рядок або до максимальної кількості мінус один символ;
* якщо **fgets()** зустрічає символ нового рядка, вона дописує його у відведений буфер на відміну від **gets()**, яка його просто відкидає, але **fputs()** не додає символ переходу на новий рядок у вихідні дані, на відміну від **puts()**.

Третім аргументом потрібно вказати файл, звідки потрібно проводити читання. Для читання символів з клавіатури вказується стандартний ідентифікатор **stdin** (**stdout** для **fputs()**).

Вивести значення рядка на екран можна за допомогою команд:

**puts(<назва рядка>); fputs(<назва рядка>, stdout);** **printf("%s", <назва рядка>);**

Приклади:

|  |  |
| --- | --- |
| **char line[81]; while(gets(line)) puts(line);** | **char line[81];**  **while(fgets(line, 81, stdin)) fputs(line, stdout);** |

Обидві наведені ділянки коду будуть виконуватися, поки не зустрінеться символ **EOF**, або поки програма не завершиться аварійно.

Крім того, стандарт С11 пропонує безпечні варіанти попередніх функцій: **scanf\_s(), printf\_s(), gets\_s()**.

Посимвольно вводити чи виводити елементи рядка можна за допомогою команд циклу **for** або **while** і використанням функцій бібліотеки **<cstdio>** **getchar()** та **putchar()**, чи їх файлових еквівалентів **getc()** та **putc()**.

Наприклад:

**char str[256]; int n;**

**for(n = 0; (str[n] = getchar()) != '\n'; ++n);**

**str[n] = '\0';**

**for(n = 0; str[n]; ++n) putchar(str[n]);**

Крім того, можна зустріти використання функцій **fgetc()** та **fputc()**. Вони еквівалентні до **getc()** та **putc()** і єдина різниця полягає в тому, що у деяких компіляторах останні можуть бути реалізовані як **#define** макроси.

5.2.2. Функції для опрацювання рядків у стилі С

Для опрацювання масивів символів у мові C++ є стандартні функції, які описані у бібліотеці **<сstring>**. Розглянемо деякі з них [11]:

**size\_t strlen(const char \*str);**

повертає кількість символів у рядку.

**char \*strcat(char \*dst, const char \*src);**

команда з'єднання рядків **dst** та **src**, результат записується в **dst,** повертає **dst**;

**char \*strncat(char \*dst, const char \*src, size\_t num);**

до змінної **dst** додає перших **num** символів рядка **src**;

**char \*strcpy(char \*dst, const char \*src);**

копіює символи з рядка **src** в рядок **dst**;

**char \*strncpy(char \*dst,const char \*src, size\_t num );**

копіює перших **num** символів рядка **src** в рядок **dst**;

**char \*strchr(const char \*str, int ch);**

визначає перше входження деякого символу **ch** у рядок **str**, повертає рядок, який починається від першого входження заданого символу до кінця рядка;

**char \*strrchr(const char \*str, int ch);**

визначає останнє входження заданого символу **ch** у рядок **str**;

**size\_t strspn(const char \*str1, const char \*str2);**

визначає номер першого символу, який входить у рядок **str1**, але не входить у рядок **str2**;

**char \*strstr(const char \*str1, const char \*str2 );**

визначає в рядку **str1** підрядок, що починається з першого входження рядка **str2** у рядок **str1**;

**char \*strtok(char \*str, const char \*delimiters);**

функція-токенайзер – розбиває рядок **str**, на окремі частини (лексеми), що розділені одним з символів-роздільників **delimiters**, якщо частина знайдена, повертає вказівник на неї, у іншому випадку повертає нуль. Функція зазвичай викликається кілька разів. При першому виклику **str** повинен відповідати потрібному рядку. При наступних викликах **str** повинен відповідати нулю, а пошук автоматично продовжуватиметься у початковому рядку. Нприклад:

**char str[] ="- This, a sample string.";**

**char \* pch;**

**printf ("Splitting string \"%s\"\n"**

**"into tokens:\n",str);**

**pch = strtok (str," ,.-");**

**while (pch != NULL)**

**{**

**printf ("%s\n", pch);**

**pch = strtok (NULL, " ,.-");**

**}**

У результаті отримаємо:

**Splitting string "- This, a sample string." into tokens: This a sample string**

Слід зауважити, що функція **strtok()** модифікує рядок – всі роздільники заміняються на нуль символи.

Наступні функції не є частиною стандартної бібліотеки і можуть бути відсутні у деяких компіляторах:

**сhar \*strnset(char \*str, int ch, size\_t count);**

вставляє **count** разів заданий символ **ch** перед рядком **str**;

**char \*strupr(char \*str);**

перетворює усі малі літери рядка у великі;

**char \*strlwr(char \*str);**

перетворює усі великі літери рядка у малі;

|  |  |  |
| --- | --- | --- |
|  | **char \*strrev(char \*str);** |  |

записує рядок у зворотному порядку.

Розглянемо результати застосування функцій до таких змінних:

**char lviv[] = "Lviv polytechnic",**

**un[30] = "NU ",**

**r1[30] = " ",**

**r2[30] = "";**

**char \*p; int n;**

|  |  |
| --- | --- |
| Застосування функцій | Результат |
| **n = strlen(lviv);** | **n = 16** |
| **strcat(un, lviv);** | **un = "NU Lviv polytechnic"** |
| **strncat(r1, lviv, 8);** | **r1 = " Lviv pol"** |
| **strcpy(r1, lviv);** | **r1 = "Lviv polytechnic"** |
| **strncpy(r2, lviv, 8);** | **r2 = "Lviv pol"** |
| **p = strchr(lviv, 'p');** | **p = "polytechnic"** |
| **p = strrchr(lviv, 'i');** | **p = "ic"** |
| **n = strspn(lviv, "Lviv");** | **n = 4** |
| **p = strstr(lviv, "tech");** | **p = "technic"** |
| **p = strtok(lviv, "iv");** | **p = "L"** |

У бібліотеці **<сstdlib>** є стандартні функції перетворення типів даних. Зокрема, функція **atoi(r1)** перетворює рядок символів **r1** у дане цілого типу **int**, а функція **itoa(<числове дане>, r1, <система числення>)** – дане цілого типу **int** у рядок **r1**. Для перетворення даних типу **double** у рядок символів визначена функція **gcvt(<числове дане>, <кількість знаків у числі>, r1)**, а обернену дію виконує функція **strtod()**.

Розглянемо результати дії цих функцій для оголошених нижче змінних:

**int n;**

**double f;**

**char r1[5], \*p;**

|  |  |
| --- | --- |
| Застосування функції | Результат |
| **n = atoi("12");** | **n = 12** |
| **itoa(12, r1, 10);** | **r1 = "12"** |
| **gcvt(-3.14, 4, r1);** | **r1 = "-3.14"** |
| **f = strtod("-3.1415", &p);** | **f = "-3.141500"** |

Рядки символів можна порівнювати між собою. Два рядки порівнюють зліва направо посимвольно, причому **'А' < 'В'**, **'В' < 'С'** тощо. "Більшим" вважається символ, який розміщений в алфавіті далі (він має більший номер у таблиці кодів ASCII). Для порівняння рядків у модулі **<cstring>** надана функція:

**int strcmp(const char \*str1, const char \*str2);**

порівнює рядки символів **str1** і **str2** з урахуванням регістра для латинського алфавіту. Результатом виконання є від'ємне число (якщо рядок **str1** менший від рядка **str2**), **0** (якщо рядки однакові) або додатне число (рядок **str1** більший за рядок **str2**).

Розглянемо результат дії функцій

|  |  |
| --- | --- |
| Функція | Результат |
| **n = strcmp("Spring", "spring");** | **n = -1** |
| **n = strcmp("spring", "Spring");** | **n = 1** |
| **n = strcmp("Spring", "Spring");** | **n = 0** |

Нехай задано рядок **"One two three"**. Визначити довжину рядка. Вивести на екран друге слово цього рядка:

|  |
| --- |
| **#include <cstdio> #include <сstring>**  **int main()**  **{**  **char r1[ ] = "One two three";**  **printf("%s\n", r1);**  **printf("The length of string = %d\n",**  **strlen(r1));** |

**char \*p= strtok(r1, " ");**

**p = strtok(NULL, " ");**

**printf("%s\n", p);**

**return 0;**

**}**

Результат:

**One two three**

**The length of string = 13 two**

Подамо ще один спосіб розв'язання задачі, в якому рядок розглядається як масив символів і функції для роботи з рядками не використовуються:

|  |
| --- |
| **#include <cstdio>**  **int main()**  **{**  **char r1[ ] = "One two three";**  **printf("%s\n",r1);**  **int len; for(len=0;r1[len];++len);**  **printf("The length of string = %d\n",len);**  **int n;**  **for(n=0;r1[n]!=' ';++n);**  **for(++n;r1[n]!=' ';++n)putchar(r1[n]);**  **return 0;**  **}** |

Результат:

**One two three**

**The length of string = 13 two**

Нехай необхідно ввести п'ять назв столиць європейських країн та впорядкувати їх за алфавітом:

|  |
| --- |
| **#include <cstdio> #include <сstring>**  **int main()**  **{**  **char str[5][256];**  **for(int i=0; i<5; ++i)**  **scanf("%s", str[i]);**  **for(int i=0; i<5; ++i)**  **for(int j=0; j<4; ++j)**  **if(strcmp(str[j], str[j+1]) > 0)**  **{** |

**char tmp[256];**

**strcpy(tmp, str[j]);**

**strcpy(str[j], str[j+1]);**

**strcpy(str[j+1], tmp);**

**}**

**for(int i=0; i<5; ++i) puts(str[i]);**

**return 0;**

**}**

Результат:

|  |  |
| --- | --- |
| Вхідні дані | Вихідні дані |
| **Kiev**  **Warsaw**  **London**  **Paris Rome** | **Kiev**  **London**  **Paris**  **Rome**  **Warsaw** |

Застосування функцій **getchar()** і **putchar()** ілюструється наступною програмою. Вона вводить символи із клавіатури і виводить їх у протилежному регістрі, тобто великі букви стають малими, і навпаки. Щоб зупинити виконання програми, необхідно ввести **Ctrl+Z**:

|  |
| --- |
| **#include <cstdio>**  **#include <cctype>**  **int main(){**  **char ch;**  **puts("Please input text. Press Ctrl+Z to finish");**  **while((ch=getchar())!= EOF)**  **{**  **if(islower(ch)) ch = toupper(ch);**  **else ch = tolower(ch);**  **putchar(ch);**  **}**  **return 0;**  **}** |

Результат:

**Please input text. Press Ctrl+Z to finish abcdEFGH ABCDefgh**

**^Z**

У наведеній програмі для роботи з верхнім та нижнім регістром використовуються стандартні функції бібліотеки **<cctype>**: **islower()**, **toupper()** та **tolower()**. Їх назви говорять самі за себе.

Функція **getchar()** може породити кілька проблем. Звичайно, ця функція поміщає вхідні дані в буфер, поки не буде натиснута клавіша <**ENTER**>. Такий спосіб називається буферизованным вводом (*lіne-buffered іnput*). Для того щоб дані, що були введені, дійсно передалися програмі, необхідно натиснути клавішу <**ENTER**>. Крім того, при кожному виклику функція **getchar()** вводить символи по одному, послідовно розміщаючи їх у черзі. Якщо програма використовує інтерактивний діалог, таке гальмування стає дратівним фактором. Незважаючи на те що стандарт мови C дає змогу зробити функцію **getchar()** інтерактивною, ця можливість використовується рідко. Отже, якщо раптом виявиться, що наведена вище програма працює не так, як очікувалося, причина буде очевидна.

Функція **getchar()**, реалізована компілятором, може не відповідати вимогам інтерактивного середовища. У цьому випадку можна використовувати інші функції, що дають змогу зчитувати символи із клавіатури. У стандарті мови C не передбачено жодної функції, що гарантує інтерактивний ввід, але на практиці цей недолік заповнюється компіляторами.

Найбільш відомими альтернативами є функції **getch()** і **getche()**. Їхні прототипи виглядають так.

**іnt getch(voіd);**

**іnt getche(voіd);**

У деяких компіляторах перед іменами цих функцій ставиться знак підкреслення. Наприклад, у компіляторі Mіcrosoft Vіsual C++ ці функції називаються **\_getch()** і **\_getche()** відповідно.

Після натискання клавіші функція **getch()** негайно повертає результат, введений символ на екрані не відображається. Функція **getche()** аналогічна функції **getch()**, за одним виключенням: введений символ відображається на екрані (*get character echo*). В інтерактивних програмах замість функції **getchar()** найчастіше використовуються функції **getch()** і **getche()**. Однак, якщо компілятор не підтримує ці функції, або функція **getchar()**, реалізована ним, повністю відповідає вимогам інтерактивного середовища, варто застосовувати саме її.

Перепишемо попередню програму, заміняючи функцію **getchar()** функцією **getch()**. Також замінимо умову виходу з програми, оскільки тепер неможливо розпізнати комбінацію **Ctrl+Z**:

**#include <cstdio>**

**#include <cctype>**

**int main()**

**{**

|  |
| --- |
| **char ch;**  **puts("Please input text. Press '.' to finish");**  **while((ch=getch())!= '.')**  **{**  **if(islower(ch)) ch = toupper(ch);**  **else ch = tolower(ch);**  **putch(ch);**  **}**  **return 0;**  **}** |

Результат:

|  |  |
| --- | --- |
| Вхідні дані | Вихідні дані |
| **abcdEFGH.** | **ABCDefgh** |

У ході виконання цієї програми при кожному натисканні клавіші відповідний символ буде негайно переданий програмі і відображений на екрані. Ввід не буферизується.

Наступна програма демонструє застосування декількох основних консольних функцій вводу-виводу при роботі з комп'ютерним словником. Програма пропонує користувачеві ввести слово, а потім перевіряє, чи міститься це слово в списку відомих:

|  |
| --- |
| **#include <cstdio>**  **#include <cstring> #include <conio.h> int main()**  **{**  **char dic[][20] = {**  **"car","ship","train","bike","plane"};**  **char word[20], ch;**  **do**  **{**  **puts("Please input a word");**  **gets(word);**  **int n;**  **for(n=0; n<5; ++n)**  **{**  **if(!strcmp(dic[n], word))**  **{**  **puts("Word is found!");**  **break;**  **}** |

**}**

**if(n==5)puts("There is no such word");**

**puts("Continue? [y]/[n]");**

**} while((ch = getch()) != 'n');**

**return 0;**

**}**

Результати:

**Please input a word fox**

**There is no such word**

**Continue? [y]/[n] Please input a word ship**

**Word is found!**

**Continue? [y]/[n]**

Нехай необхідно ввести текст, визначити, скільки разів в тексті зустрічається кожна буква латинського алфавіту:

|  |
| --- |
| **#include <cstdio> #include <cstring>**  **int main()**  **{**  **int letters[26], length = 0;**  **char text[1024], ch;**  **for(int i=0; i<26; ++i) letters[i] = 0;**  **puts("Please input text. Press Ctrl+Z to stop");**  **while((ch = getchar()) != EOF)**  **{**  **if((ch >= 'a')&&(ch <= 'z'))**  **letters[ch - 'a']++;**  **else if((ch >= 'A') && (ch <= 'Z'))**  **letters[ch -'A']++;**  **++length;**  **}**  **puts("Letters frequency:");**  **for(int i=0; i<26; i += 2)**  **printf("%c: %7.3lf%%\t%c: %7.3lf%%\n",**  **i + 'A', 100.0 \* letters[i] / length,**  **i+'A' + 1, 100.0 \* letters[i+1] / length);**  **return 0;**  **}** |

Результати:

|  |
| --- |
| **Please input text. Press Ctrl+Z to stop**  **Time is money. (Edward George Bulwer-Lytton)**  **^Z**  **Letters frequency:**  **A: 2.222% B: 2.222%**  **C: 0.000% D: 4.444%**  **E: 13.333% F: 0.000%**  **G: 4.444% H: 0.000%**  **I: 4.444% J: 0.000%**  **K: 0.000% L: 4.444%**  **M: 4.444% N: 4.444%**  **O: 6.667% P: 0.000%**  **Q: 0.000% R: 6.667%**  **S: 2.222% T: 6.667%**  **U: 2.222% V: 0.000%**  **W: 4.444% X: 0.000% Y: 4.444% Z: 0.000%** |

При написанні програм звичайно необхідно робити різні маніпуляції з рядками: копіювати їх і переносити з одного місця пам'яті в інше, перевіряти наявність у рядках певної послідовності символів, об’єднувати, зменшувати рядки і т. п. Мова С містить багату колекцію функцій для роботи з рядками. Кілька загальних зауважень щодо використання функцій бібліотеки:

* якщо у функції виконується перенос символів рядка з одного місця в інше, то для рядка призначення потрібно попередньо зарезервувати місце в пам'яті;
* копіювання рядків з використанням неініціалізованого вказівника – одна з найпоширеніших помилок програмування, навіть у

досвідчених програмістів;

* працюючи з рядками, необхідно обов'язково звертати увагу на попередження компілятора типу "*Підозріле перетворення вказівника*" (*Suspіcіous poіnter conversіon*) або "*Використання вказівника до ініціалізації*" (*Possіble use of...before deffіnіtіon*). При виділенні місця для рядка-призначення варто передбачити місце для нуль-термінатора (**'\0'**).

**5.2.3. Рядки в C++ (клас String)**

**Основи роботи**

У мові C++ для зручної роботи з рядками є клас **String**, для якого потрібно включити заголовний файл string.

Рядки можна оголошувати та ініціалізувати одночасно:

**string S1, S2 = "Hello";**

Рядок S1 буде порожнім, рядок **S2** буде довжиною 5 символів.

До окремих рядкових символів ви можете посилатися по індексу так, ніби це масив або елемент C-рядок. Наприклад, **S[0]** є першим символом рядка.

Для того, щоб дізнатися довжину рядка, можна скористатися методом **size()** рядка. Наприклад, останнім символом рядка **S** є **S[S.size() - 1].**

**Конструктори рядків**

Створити рядки можна за допомогою таких конструкторів:

* **string()** - конструктор за замовчуванням (без параметрів) створює порожній рядок.
* **string(string & S)** - копія рядка **S**
* **string(size\_t n, char c)** - повторення символу **c** задану кількість разів.
* **string(size\_t c)** - рядок з одного символу **c.**
* **string(string & S, size\_t start, size\_t len)** - рядок, що містить не більше ніж **len** символів даного рядка **S**, від символу **start.**

Конструктори можна викликати явно, наприклад, так:

**S += string(10, 'z');**

У цьому прикладі явно викликається конструктор **string** , щоб створити рядок із 10 символів'z'.

Неявно конструктор викликається при оголошенні рядка з додатковими параметрами. Наприклад, так:

**string S(10, 'z');**

**Рядковий ввід-вивід**

Рядок виводиться так само, як і числові значення:

**cout << S;**

Щоб прочитати рядок, ви можете скористатися командою ">>" для об'єкта cin:

**cin >> S;**

У цьому випадку читається рядок непробільних символів, пропускаючи пробіли та символи кінця рядків. Це корисно для розбиття тексту на слова або для читання даних до кінця файла за допомогою while (cin >> S).

Рядки можна зчитувати доки не з'явиться символ кінця рядка з функції getline. Сам символ кінця рядка читається з вхідного потоку, але не додається до рядка:

**getline(cin S);**

**Арифметичні оператори**

Над рядками можна виконати наступні арифметичні дії:

**=** –присвоїти значення.

**+=** – додати ще один рядок або символ в кінець рядка.

**+** – конкатенація двох рядків, конкатенація рядка і символу.

**==, !=** – посимвольне порівняння.

**<, >, <=, >=** – лексикографічне порівняння.

Тобто можна скопіювати вміст одного рядка в інший за допомогою операції **S1 = S2**, порівняти два рядки на рівність за допомогою **S1 == S2**, порівняти рядки в лексикографічному порядку за допомогою **S1 < S2** або зробити додавання (конкатенацію) двох рядків у вигляді **S = S1 + S2**.

**Рядкові методи**

Рядки мають різні методи, багато з яких можуть бути використані декількома пособами (з різним набором параметрів). Розглянемо ці методи докладніше.

**size**

Метод **size()** повертає довжину рядка. Значення, що повертається, є беззнаковим типом (як і у всіх випадках, коли функція повертає значення, рівне довжині рядка або індексу елемента - ці значення є беззнаковими). Тому потрібно акуратно виконати операцію віднімання зі значення, яке повертає **size().** Наприклад, було б помилкою написати цикл, який перебирає всі символи рядка, крім останнього у формі **for (int i = 0; i < S.size() - 1; ++i).**

Крім того, рядки мають метод **length()**, який також повертає довжину рядка.

**resize**

**S.resize(n)** - Змінює довжину рядка, нова довжина рядка стає **n**. При цьому рядок може як зменшуватися, так і збільшуватися. Якщо викликати у вигляді **S.resize(n, c),** де **c** - символ, то при збільшенні довжини рядка додані символи будуть рівні **c**.

**clear**

**S.clear()** - очищає рядок, рядок стає порожнім.

**empty**

**S.empty()** - Повертає **true**, якщо рядок порожній, **false**, якщо він не порожній.

**push\_back**

**S.push\_back(c)**- додає символ **c** в кінець рядка, що викликається єдиним параметром типу **char**.

**append**

Додає кілька символів у кінець рядка, інший рядок або фрагмент іншого рядка. Має багато способів виклику.

**S.append(n, c)** - додає **n** однакових символів, рівних **c**. **n** має цілочисельний тип, **c** - char.

**S.append(T)** - додає вміст рядка **T** в кінець рядка **S. T** може бути об'єктом класу string або **C**-рядком.

**S.append(T, pos, count)** - додає символи рядка **T** в кінець рядка **S**, починаючи з символу з індексом pos кількістю count.

**erase**

**S.erase(pos)**- видаляє з рядка **S** з символа з індексом **pos** і до кінця рядка.

**S.erase(pos, count)** - видаляє з рядка **S**  з символа з індексом pos кількістю count або до кінця рядка, якщо **pos + count > S.size().**

**insert**

Вставляє кілька символів, ще один рядок або фрагмент іншого рядка в середину рядка. Методи виклику подібні до методу додавання, за винятком того, що першим параметром є значення i, позиція, в якій вставляються символи. Перший вставлений символ матиме індекс i, а всі символи, які раніше мали індекс i і більше, будуть зсунуті вправо.

**S.insert(i, n, c)** - вставити n однакових символів, рівних с. n має цілочисельний тип, c - char.

**S.insert(i, T) -** вставити вміст рядка **T**. **T** може бути об'єктом **string** або **C**-рядком.

**S.insert(i, T, pos, count)** - вставити символи рядка **T** починаючи з символу з індексом **pos** кількіст **count**.

**substr**

**S.substr(pos)** - повертає підрядок цього рядка, починаючи з символу з індексом pos і до кінця рядка.

**S.substr(pos, count)** - повертає підрядок вказаного рядка, починаючи з с індексом **pos** кількістю **count** або до кінця рядка, якщо **pos +** **count > S.size().**

**replace**

Замінює фрагмент рядка кількома однаковими символами, іншим рядком або фрагментом іншого рядка. Методи виклику подібні до методу **append**, за винятком того, що першими двома параметрами є два числа: **pos** і **count.** З цього рядка видаляється кількість символів, починаючи з символу **pos**, а на їх місце вставляються нові символи.

**S.replace(pos, count, n, c)** - вставити n однакових символів, що дорівнюють с. n має цілочисельний тип, **c** - char.

**S.replace(pos, count, T)** - вставити вміст рядка **T**. **T** може бути об'єктом класу **string** або **C**-рядком.

**S.replace(pos, count, T, pos2, count2)**– вставити символи рядка **T** починаючи з символу с індексом pos кількістю count.

**find**

Шукає перше входження іншого рядка **str**. Повертається номер першого символу, починаючи з підрядка, що дорівнює рядку **str**. Якщо цей рядок не знайдено, буде повернуто константу **string::npos** (яка дорівнює -1, але є беззнаковою, тобто насправді є великим недодатним числом).

Якщо задано значення **pos**, тоді пошук починається з позиції **pos,** тобто значення, що повертається, буде не менше pos. Якщо значення **pos** не вказано, вважається, що вона дорівнює 0 - пошук виконується від початку рядка.

**S.find(str, pos = 0)** - Шукаємо перший запис рядка str, починаючи з позиції pos. Якщо pos не вказано, то починаючи з початку рядка **S**.

**S.find(str, pos, n)** - Шукаємо в цьому рядку підрядок, який дорівнює першим n символам рядка **str**. Значення **pos** має бути встановлене.

**rfind**

Шукає останнє входження підрядка ("правий" пошук). Методи виклику аналогічні методам виклику методу **find.**

**find\_first\_of**

Шукає перше входження будь-якого з символів рядка **str.** Повертається номер цього символу або значення **string::npos**.

Якщо встановлено значення pos, тоді пошук починається з позиції **pos**, тобто значення, що повертається, буде не менше **pos.** Якщо значення **pos** не вказано, вважається, що дорівнює 0 - пошук виконується від початку рядка.

**S.find\_first\_of(str, pos = 0)** - пошук першого входження будь-якого символу рядка str починаючи з позиції **pos.** Якщо **pos** не задано - то починаючи з початку рядка **S**.

**find\_last\_of**

Шукає останнє входження будь-якого з символів рядка **str.** Методи виклику та значення, що повертається, подібні до методу **find\_first\_of**.

**find\_first\_not\_of**

Шукає перше входження символу, відмінного від рядка **str**. Методи виклику та значення, що повертається, такі самі, як і **find\_first\_of**.

**find\_last\_not\_of**

Шукає останнє входження символу, відмінного від рядка **str**. Методи виклику та значення, що повертається, подібні до методу **find\_first\_of.**

**c\_str**

Повертає вказівник в область пам'яті, де зберігаються символи рядка, повертає значення типу **char\***. Значення, що повертається, можна розглядати як C-рядок і використовувати у функціях, яким потрібно отримати C-рядок на вхід.

5.2.4. Таблиця ASCII символів

Символи зберігаються в пам'яті як числові коди. Найбільш часто застосовується система кодування *ASCII* (*American Standard Code for Information* Interchange) – система кодів, у якій числа від 0 до 127 включно поставлені у відповідність літерам, цифрам і символам пунктуації. Існують і інші розширені системи кодування. ASCII можна розглядати як 7-бітну схему кодування підмножини Unicode/UCS. UTF-8 сумісний з ASCII для кодів менше 128.

Мова С++ дає змогу представити більшість символів безпосередньо шляхом їх укладення в одинарні лапки, наприклад, **'А'** для символу **А**. Крім того, символ можна представити з використанням його вісімкового або шістнадцяткового коду, перед яким повинен знаходитися зворотній слеш, наприклад, **'\012'** та **'\0xa'** відповідають символу переходу на новий рядок **'\n'**. Керуючі послідовності подібного роду також можуть бути частиною рядка, скажімо, такий: **"Please \012welcome!"**.

Перші 32 символи задумані для керування периферійними пристоями і не

призначені для відображення на екран (символ **^** відповідає клавіші **Ctrl**):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Код | | Назва | | Керуюча послідовність | | Символ | | Опис | |
| **0x00** | | **NUL** | | **\0** | | **^@** | | Null character | |
| **0x01** | | **SOH** | |  | | **^A** | | Start of Heading | |
| **0x02** | | **STX** | |  | | **^B** | | Start of Text | |
| **0x03** | | **ETX** | |  | | **^C** | | End of Text | |
| **0x04** | | **EOT** | |  | | **^D** | | End of Transmission | |
| **0x05** | | **ENQ** | |  | | **^E** | | Enquiry | |
| **0x06** | | **ACK** | |  | | **^F** | | Acknowledgment | |
| **0x07** | | **BEL** | | **\a** | | **^G** | | Bell | |
| **0x08** | | **BS** | |  | | **^H** | | Back Space | |
| **0x09** | | **HT** | | **\t** | | **^I** | | Horizontal Tab | |
| **0x0a** | | **LF** | | **\n** | | **^J** | | Line Feed | |
| **0x0b** | | **VT** | | **\v** | | **^K** | | Vertical Tab | |
| **0x0c** | | **FF** | | **\f** | | **^L** | | Form Feed | |
| **0x0d** | | **CR** | | **\r** | | **^M** | | Carriage Return | |
| **0x0e** | | **SO** | |  | | **^N** | | Shift Out / X-On | |
| **0x0f** | | **SI** | |  | | **^O** | | Shift In / X-Off | |
| **0x10** | | **DLE** | |  | | **^P** | | Data Line Escape | |
| **0x11** | | **DC1** | |  | | **^Q** | | Device Control 1 (XON) | |
| **0x12** | | **DC2** | |  | | **^R** | | Device Control 2 | |
| **0x13** | | **DC3** | |  | | **^S** | | Device Control 3 (XOFF) | |
| **0x14** | | **DC4** | |  | | **^T** | | Device Control 4 | |
| **0x15** | | **NAK** | |  | | **^U** | | Negative Acknowledgement | |
| **0x16** | | **SYN** | |  | | **^V** | | Synchronous Idle | |
| **0x17** | | **ETB** | |  | | **^W** | | End of Transmit Block | |
| **0x18** | | **CAN** | |  | | **^X** | | Cancel | |
| Код | | Назва | | Керуюча послідовність | | Символ | | Опис | |
| **0x19** | | **EM** | |  | | **^Y** | | End of Medium | |
| **0x1a** | | **SUB** | |  | | **^Z** | | Substitute | |
| **0x1b** | | **ESC** | | **\e** | | **^[** | | Escape | |
| **0x1c** | | **FS** | |  | | **^\** | | File Separator | |
| **0x1d** | | **GS** | |  | | **^]** | | Group Separator | |
| **0x1e** | | **RS** | |  | | **^^** | | Record Separator | |
| **0x1f** | | **US** | |  | | **^\_** | | Unit Separator | |

Символи з 32-го по 126 є символами, що відображаються на екрані чи друкуються на принтері. Символ 32 це пробіл, символ 127 відповідає команді стирання DEL.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **+0 +1 +2 +3 +4 +5 +6 +7** | | | | | | | |
| **0x20** |  | **!** | **“** | **#** | **$** | **%** | **&** | **'** |
| **0x28** | **(** | **)** | **\*** | **+** | **,** | **-** | **.** | **/** |
| **0x30** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **0x38** | **8** | **9** | **:** | **;** | **<** | **=** | **>** | **?** |
| **0x40** | **@** | **A** | **B** | **C** | **D** | **E** | **F** | **G** |
| **0x48** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** |
| **0x50** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** |
| **0x58** | **X** | **Y** | **Z** | **[** | **\** | **]** | **^** | **\_** |
| **0x60** | **`** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| **0x68** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** |
| **0x70** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** |
| **0x78** | **x** | **y** | **z** | **{** | **|** | **}** | **~** |  |

# 5.3. Контрольні запитання

1. Яким чином відбувається оголошення та ініціалізація рядків символів (стрічок)?
2. Які функції для роботи з стрічками ви знаєте?
3. Назвіть операції порівняння стрічок, коротко поясніть результати їх дії.

# 5.4. Лабораторне завдання

1. Навчитися використовувати символьні масиви для розв’язання задач роботи зі стрічками.
2. Одержати індивідуальне завдання.
3. Побудувати блок-схеми алгоритмів відповідно до завдання.
4. Скласти програми на алгоритмічній мові C згідно завдання.
5. Відлагодити програми, виконати обчислення, проаналізувати отримані результати.

**5.5. Зміст звіту** 1. Титульний аркуш.

1. Мета роботи.
2. Індивідуальне завдання.
3. Блок-схема алгоритмів у відповідності до завдання.
4. Тексти програм у відповідності до завдання.
5. Результати обчислень.
6. Аналіз результатів, висновки.

# 5.6. Індивідуальні завдання

Завдання 1

Ввести з клавіатури своє прізвище, ім'я та по батькові як одне текстове дане. Виконати описані нижче дії. Результати вивести на екран.

1. Вивести ім'я та кількість букв у третьому слові.
2. Визначити скільки букв **'a'** у прізвищі.
3. Вивести ініціали.
4. Вивести прізвище та ініціали.
5. Вивести ім'я та кількість букв у прізвищі.
6. Визначити скільки букв **'о'** є в імені.
7. Вивести найдовше слово.
8. Вилучити усі букви **'a'** та **'о'** з прізвища.
9. Вивести ім'я у стовпчик.
10. Продублювати всі букви в імені.
11. Вивести ім'я у зворотному порядку.
12. Вивести прізвище у стовпчик у зворотньому порядку.
13. Вивести найкоротше слово.
14. Потроїти голосні букви у імені?
15. Вивести ініціали та прізвище у зворотньому порядку.
16. Вивести кількість букв у імені та прізвищі.

Завдання 2

Символьні рядки S1 та S2 довжиною до 100 символів вводити з клавіатури. Виконати описані нижче дії. Результати вивести на екран.

1. Підрахувати кількість приголосних у S1 та S2, що з обох боків оточені цифрами.
2. Дописати у S2 ті символи з S1, які не зустрічаються у S2.
3. Знайти у S1 найдовшу послідовність літер, що розташовані по зростанню, а в S2 – по спаданню (за алфавітом).
4. Порахувати кількість цифр з S2, які зустрічаються в S1.
5. Змінити порядок символів у S1 на зворотній. Замінити усі маленькі приголосні S2 на великі.
6. Видалити з S1 усі входження S2.
7. Записати, відокремивши пробілами, у середину S1 всі символи з S2, що не є буквами, або цифрами.
8. Відсортувати по зростанню S1 та по спаданню S2 (за алфавітом).
9. Вивести кількість приголосних з S1, які зустрічаються в S2.
10. Дописати у початок S1 символи, які є присутніми в обох рядках.
11. Розділити символи S1 на три групи: голосні, приголосні, цифри. Групи розділити пробілами та відсортувати їх по зростанню (за алфавітом). Аналогічно для S2 – по спаданню.
12. Видалити усі пропуски з S1 та дописати їх у початок S2.
13. Підрахувати кількість входжень S2 у S1, та дописати їх у початок S2 в зворотньому порядку.
14. Підрахувати кількість голосних у S1, що з обох боків оточені приголосними. Аналогічно для S2 – кількість приголосних, оточених голосними.
15. Знайти у S1 найдовшу послідовність цифр, а в S2 – голосних літер.
16. Подвоїти пробіли між словами в S1 та потроїти в S2.