**Лекція 7. Символьні та рядкові величини. Функції для роботи з символьними змінними.**

**Символ — найменша складова тексту**

**Символьна величина — це величина типу char для збереження (цілочисельного) коду символу**

**Рядки — упорядковані послідовності символів, що використовують для зберігання і представлення текстової інформації**

**Як один з варіантів, рядок у мові С++ подають масивом елементів типу char, що закінчуються нуль – термінатором  '\0'**

**Найпростіший варіант створювати і використовувати символьні масиви для  роботи з рядками, проте такий масив (типу char) не завжди можна вважати  рядком. Справа з тому, що рядок повинен містити нульовий байт /0, він символізує кінець**

В С++ є два підходи: використовувати рядкову змінну як масив символів або як об‘єкт класу String. Перший підхід – спадок мови С (не останній в мові С++). Розглянемо різницю між С (яка є складовою С++) та С++.

C — це **системна мова програмування** **загального призначення**. В цьому її унікальність. C частіше за всі решту мов використовується для програмування системного забезпечення, зокрема, програми самого комп‘ютера. В цьому причина, що C вважається однією з найсложніших мов програмування в світі. C можна назвати прародителем мови C++, остання успадкувала від неї основні особливості.

Щодо C++ - це **мова програмування загального призначення**.

Головною різницею між C++ та C є те, що С++ (переважно) є **об‘єктно-орієнтованою мовою**, тоді як C є **процедурним**. Оскільки C в основному використовується для розробки комп‘ютерних систем, то ця різниця не є критичною, але якщо розглядати прикладну розробку C++ буде мати переваги перед C. Причина полягає в тому, що якщо мова є об‘єктно-орієнтованою (повністю або частково), вона вважається **адаптованою.** C++ орієнтований на **об‘єкти** та **дані,** ніж **дії** та **логіку** (це склалось історично). Об‘єктно-орієнтоване програмування дозволяє мові сфокусуватися на самому об‘єкті, ігноруючи менш важливі деталі в процесі.

У переважній кількості випадків C++ дуже схожий на C. Обидва легкі, маютьт ручне управлення пам‘яттю і можуть використовуватися для програмування практично всього.

Критерії порівняння C та C++ - ключові аспекти: **швидкодія, популярність** та можлива **зарплата.  
 Про різницю.**

**Як відомо, дуже багато С++ запозичила у мови С. Ці запозичення забезпечили С++ потужними засобами низького рівня, що дозволяють розв’язувати завдання системного програмування. Але С++ відрізняється від С в першу чергу різним ступенем уваги до типів і структур даних. Це пов’язано з появою понять класу, похідного класу і віртуальної функції. Це дає С++ більш ефективні можливості для контролю типів даних і забезпечує модульність програми**

На відміну від ООП, процедурне програмування орієнтоване на послідовність виконуваних дій. Воно використовує підпрограми, звані процедурами або функціями, для поділу коду на більш дрібні блоки та повторного використання коду.

Розглянемо це на прикладі таблиці:

|  |  |  |
| --- | --- | --- |
| **Характеристика** | **Процедурне програмування** | **Об’єктно-орієнтоване програмування** |
| Основні концепції | Процедури та функції, дані поділені на змінні | Класи, об‘єкти, спадкування, поліморфізм |
| Приклади мов програмування | Сі, Паскаль, Фортран | Java, C++, Python, Ruby |
| Організація коду | Функції та процедури розділені за завданнями | Об’єкти містять методи та дані |
| Використання даних | Здебільшого використовуються глобальні змінні | Дані інкапсульовані в об’єктах |
| Керування пам’яттю | Ручне керування пам’яттю | Автоматичне керування пам’яттю |
| Швидкість виконання | Швидше через низький рівень абстракції | Трохи повільніше через високий рівень абстракції |
| Простота і зрозумілість коду | Простота в написанні, зрозумілість для початківців | Складніший у написанні, потребує певного досвіду |
| Розширюваність коду | Обмежена розширюваність коду через відсутність успадкування | Вища розширюваність завдяки успадкуванню та поліморфізму |

Як бачимо, процедурне програмування надає простий і зрозумілий код, швидке виконання та легкість налагодження, але воно також має обмежену гнучкість і складність розширення коду. З іншого боку, об’єктно-орієнтоване програмування дає змогу створювати складніші та гнучкіші системи, але може потребувати більш тривалого часу для розроблення і вимагати більш високого рівня абстракції та розуміння. Під час вибору між процедурним і об’єктно-орієнтованим підходами до програмування, необхідно враховувати конкретні вимоги проєкту і зробити вибір на основі їх відповідності.

Процедурне програмування відіграє ключову роль у написанні структурованого та модульного коду. Цей підхід дає змогу розбивати програми на безліч невеликих і логічно пов’язаних процедур, що значно спрощує розуміння та зміну коду в майбутньому. До того ж процедурне програмування допомагає керувати помилками в програмному коді, завдяки ретельній перевірці та налагодженню кожної процедури перед інтеграцією в основний код.

Однією з головних переваг процедурного програмування є можливість оптимізації продуктивності, тому що процедури можуть виконуватися швидше та ефективніше, ніж великі блоки коду. Однак, слід зазначити, що процедурне програмування має й недоліки, наприклад, обмежену можливість перевикористання коду та складнощі під час роботи з великими обсягами даних.

У сучасному програмуванні процедурне програмування використовується в поєднанні з іншими підходами, такими як ООП і функціональне програмування, для досягнення оптимального результату в різних завданнях програмування. Загалом процедурне програмування залишається важливою технологією в галузі програмування, забезпечуючи надійність, ефективність і структурованість коду.

Мову С проєктували з розрахунком на те, щоб використовувати у системному програмуванні. Отже, вона не вимагає додаткового часу на виконання перевірок різноманітних умов, які ніколи не відбудуться у правильно написаній програмі, а забезпечує простий, прямий доступ до адреси будь-якого об'єкта (наприклад, карти пам'яті, пристрою контролю регістрів), і її початковий код компілюється у послідовність примітивних машинних операцій. С дозволяє здійснення низки операцій, котрі часто є небажаними, а тому чимало помилок у коді не виявляє компілятор, і вони не можуть бути очевидними під час виконання. Якщо під час програмування та підтримки програмного забезпечення не дотримуватися низки суворих правил, пізніше може виникнути немало проблем зі швидкодією, стабільністю та безпекою програмного забезпечення. (Хоча, виною подібних проблем найчастіше є програмісти, бо С сама по собі забезпечує низький рівень захисту).

Рекомендую ознайомитися «10 переваг мови С++, яка і досі залишається актуальною » URL: https://dou.ua/forums/topic/42888/

**ДАНІ СИМВОЛЬНОГО ТИПУ**

Рядок (або ще кажуть "рядкова змінна") являє собою масив символів, який закінчується нуль-символом. Нагадаємо, що нуль-символ має код, що дорівнює **0**, і запис у вигляді керуючої послідовності **‘\0’**. За розташуванням нуль-символу визначається фактична довжина рядка. Кількість елементів символьного масиву складається з кількості символів у рядку плюс **1**, тому що нуль-символ також є елементом масиву.

Адреса першого символу рядка може використовуватися по-різному:

* якщо рядок застосовується при ініціюванні масиву типу **char**, адреса його першого елемента стає синонімом імені масиву. Наприклад, ідентичними є такі описи масиву:
* **char st [  ] = “Слово”;**
* **char st [6] = “Слово”;**
* **char st [6] = {‘С’ ‘л’ ‘о’ ‘в’ ‘о’ ‘\0’};**
* якщо рядок використовується для ініціювання покажчика типу **char\***, адреса першого символу рядка буде початковим значенням покажчика, наприклад:

**char \*pst = “Слово”;.**

* якщо рядок використовується у виразі, що застосовує покажчик, то компілятор підставляє у вираз рядка адресу його першого символу, наприклад:

**char \*pst;**

**pst = “Первый символ”;**

Тут **pst** одержує адресу символа **«П»** (тобто першого символу рядка).

При описі символьного масиву його ім’я — не змінна, а покажчик-константа на початок рядка, тому її не можна використовувати в деяких операціях адресної арифметики. Зокрема, не можна здійснювати операцію присвоювання вигляду:

**char st [20];**

**st = “Петренко”;** — **запис неправильний**, тому що не можна змінити значення **st**.

**Виконання дій з елементами символьного масиву** здійснюється через індекси або через покажчики. Для доступу до будь-якого символу рядка використовується індекс масиву **char**. Тобто, якщо описана змінна **char str [3];**, то третім елементом масиву можна скористатися, записавши: **str [2] або \*(str+2)**.

Для роботи з символьними масивами використовують масиви покажчиків, кожен з елементів яких містить адресу рядка масиву даних у пам’яті. Такий спосіб дозволяє зберігати дані з рваними краями, наприклад, деяку текстову інформацію. Масив з «рваними» краями схожий на двовимірну таблицю, рядки якої можуть мати різну довжину. Використання масиву покажчиків (**char \*fio[ ]**) для збереження рядків дозволяє заощаджувати пам’ять, а процес обробки рядків виконується значно швидше, бо змінюються тільки покажчики, а не вміст рядків.



Рисунок 1. Приклад масиву з "рваними" краями

**Приклад 1**. Виведення даних з масиву з "рваними" краями

**#include <iostream>**

**using namespace std;**

**main ( )**

**{ system("color F0");**

**char \*fio[ ] = { "Petrenko",**

**"Golovko",**

**"Korz",**

**"Kutz",**

**"Ushko",**

**"Plush" }; *//*** *ініціалізація масиву покажчиків*

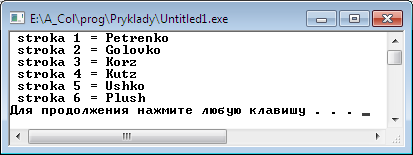
**int str;**

**for (str = 0; str <=5; str++)**

**cout << " stroka " << (str + 1) << " = " << \*(fio + str) << endl;**

**system("pause"); // для затримки екрану виведення результатів**

**}**

Результати виконання програми:  


**У процесі роботи з елементами двовимірного масиву** застосовують або індекси масиву, або індекси покажчиків. Якщо описаний список прізвищ **char°spis°[5]°[15];**, то для використання символу масиву слід записати:

**spis [і][j] або \*(spis [і] + j)**.

Аналогічно, якщо оголошений масив покажчиків **char \*str [5]**, що містить 5 елементів, кожний з яких вказує на рядок, то доступ до символу рядка можна здійснити з використанням запису **\*(str [і] + j)**.

**Введення рядків** можна здійснювати різними способами, найбільш розповсюдженими з яких є:

* введення шляхом ініціювання при оголошенні символьних масивів:

**char st 15] = “Диск”;**  
**char st [  ] = “Диск”;**  
**char \*pst = “Диск”;.**

У цьому випадку двовимірні масиви можна ініціювати по-різному, наприклад, у вигяді:

**char str [5][20] = {“Петренко И. И. “, “Головко С. С. “, . . . ,};**  
**char str [ ][20] = {“Петренко И. И. “, “Головко С. С. . . . ,};**  
**char \*pst[5] = {“Петренко И. И. “, “Головко С. С. “, . . . ,};**  
**char \*pst[ ] = {“Петренко И. И. “, “Головко С. С. “…….., };**

* використання потокового введення **сіn >>**. Здійснюється у випадку, коли рядок не містить пропусків, тому що символ пропуску є роздільником введення даних, наприклад:

**char st [5];  сіn >> st;**  
**char \*pst;  сіn >> \*pst;**  
**char str [5][20];  сіn >> str [i];**  
**char \*pst [5];  cin >> \*(pst [i]);**

* посимвольне введення за допомогою функції **get( )**, наприклад: **get (st[i]);**
* введення за допомогою функції **cin.get**:

**cin.get (str[i], size, endl);,** де **size** — кількість символів, що читаються;

* введення з використанням функції **cin.getline**:

**cin.getline (str[i], sizeof (str[i]-l));,**  
де **sizeof()** — функція визначення розміру рядка.

Виведення рядкових даних реалізується з використанням стандартного вихідного потоку **cout**:

**cout << st;**

**cout.write(st, size);** тощо.

Для потокового введення-виведення доцільно застосовувати функції **setw(w),setprecision(d), cout.width(w) і cout.precision(d).**

Введення-виведення символьних масивів можна здійснити за допомогою відповідних функцій заголовного файлу **stdio.h.**, наприклад:

* для введення рядків — **gets(st); та scanf (% s,st);**
* для виведення рядків **— puts(st); і printf(% s,st);.**

ранніх версіях С++ рядки розглядалися як символьні масиви. Для роботи з ними розроблено бібліотеку функцій **string.h,** що містить ефективні засоби для роботи з рядками. Згодом була розроблена стандартна бібліотека шаблонів **Standard TemplateLibrary (STL),** яка надає більш потужні засоби, об’єднані в клас **string**.

Для обробки символьних типів даних бібліотека функцій **string.h** має велику кількість вбудованих функцій:

* функції перевірки символів;
* функції перетворення символів;
* функції перевірки рядків;
* функції маніпулювання рядками.

Функції наводяться у вигляді списків, що згруповані за їх розташуванням у заголовних файлах.

**Прототипи, стислий опис, дія та методика застосування основних функцій обробки даних символьного типу.**

Для обробки символьних типів даних бібліотека функцій **string.h** має велику кількість вбудованих функцій, що збільшують продуктивність праці програмістів та скорочують час на розробку програм, наприклад:

* функції перевірки символів;
* функції перетворення символів;
* функції перевірки рядків;
* функції маніпулювання рядками.

Функції наводяться у вигляді списків, що згруповані за їх розташуванням у заголовних файлах. Найчастіше надаються прототипи функцій, що описують, як слід використовувати функції у програмах.

***Функції копіювання рядків:***

* **char strcpy (s, \*st);** — виконує операцію копіювання байтів рядка **st** у рядок **s**(включаючи  **“\0”**; повертає **s**), на­приклад:

**char str [50];**

**strcpy (str, “О деле суди по исходу.”);**

* **char \*strdup (const char \*str);** — виконує копіювання рядка **str** і повертає покажчик на рядок-копію, наприклад:

**char st1 = “Слово — есть поступок.”;**

**char \*st2;**

**st2 = strdup (st1);**           //копируется st1 в st2;

* **char \* strncpy (char \*st1, const char \*st2, int n);** — виконує копіювання **n** символів з рядка **st2 у st1** (рядок **stl** повинен бути більше або дорівнювати **st2**, інакше виникне помилка), наприклад:

**char st1[ ] = “Паскаль “;**

**char st2[ ] = “Привет из далека “;**

**strnсpy (st1, st2, 3);**   // st1 — “Прикаль“.

**Функції, конкатенації рядків:**

* **char \*strcat (char \*st1, const char \*st2);** — поєднує **st1 і st2** та повертає **st1**,наприклад:

**char str [100];  
strcpy (str, “Borland “);  
strcat (str, ” C++5″);,**

у результаті маємо рядок

**string = “Borland C++5”;**

* **char \*strncat (char \*st1, const char \*st2, int n);** — додає до рядка **st1 n** символів рядка **st2** і повертає знову в **st1**, наприклад:

**char st1 [90] = “Привет “;**

**char st2 [50] = “студент и студентка”;**

**strncat (st1, st2, 7);,**

у результаті маємо рядок:

**st1 = “Привет студент ” .**

***Функції порівняння рядків:***

* **int strcmp (char \*stl, char \*st2);** — порівнює рядки **st1 і st2** та повертає цілу величину, що дорівнює:

**<0 — якщо st1 < st2;**

**= 0 — якщо st1 = st2;**

**>0 — якщо st1 > st2;,**

наприклад:

**char st1[ ] = “Слово ” ;**

**char st2[ ] = “слово”;**

**int k;**

**k = strcmp (st1, st2);**        //k<0;

**int stricmp (const char \*stl, const char \*st2);** — виконує порівняння рядків, не враховуючи регістра символів; повертає цілу величину, як і функція **strcmp()**, наприклад:

**char st1[ ] = “Слово “;**

**char st2[ ] = “слово”;**

**int k;**

**k = stricmp (st1, st2);** //k = 0;

* **int strncmp (char \*stl, char \*st2, int n);** — виконує порівняння рядків із заданою кількістю символів **n у st1 і st2** і повертає цілу величину:

**<0 — якщо st1 < st2;**

**=0 — якщо st1 = st2;**

**>0 — якщо st1 > st2;**   ;

* **char \*strnicmp (char \*stl, char \*st2, int n);** — виконує порівняння рядків із заданою кількістю символів **n у st1 і st2**, незалежно від регістра, і повертає цілу величину, як і в попередньому випадку.

***Функції перетворення символів рядка:***

* **char \*strlwr (char\*st);** — перетворює символи рядка **st** верхнього регістра в символи нижнього регістра, при цьому інші символи не враховуються. Наприклад:

**char st [ ] = ” Лазерный Принтер”;**

**strlwr (st);**     // st = ” лазерный принтер” ;

* **char \*strupr (char \*st);** — перетворює символи рядка **st** нижнього регістра в символи верхнього регістра, інші символи не враховуються;
* **char \*strrev (char \*st);** — записує символи в рядку **st** у зворотному порядку (реверсує рядок), наприклад:

**char st [ ] = ” Hello”;**

**strrev (st);**   //st – ” olleH”;

* **char \*strchr (char \*st, int c);** — визначає перше входження символу **с** у рядок **st**;повертає покажчик на символ у рядку **st**, що відповідає введеному символу, наприклад:

**char st [90] = ” Borland С++5 ”**

**char \*spt;**

**spt= strchr (st, ‘+”);** — тепер покажчик **spt** вказує на підрядок **“++5”** рядка **st**;

* **char \*strrchr (char \*st, int c);** — знаходить останнє входження символу **с** у рядок **st**;якщо символ с у рядку не виявлений, повертає **0**, інакше повертає покажчик на останній символ у рядку **st**, що відповідає заданому зразку, наприклад:

**char st [80] = “Borland С++5”;**

**char \*spt;**

**spt= strrchr (st, ‘+’);** — покажчик **spt** вказує на підрядок **“+5”** рядка **st**.

***Функції пошуку підрядка в рядку:***

* **strspn (const char \*st1, const char \*st2 );** — повертає кількість символів від початку рядка **st1**, що збігаються із символами рядка **st2**, де б вони не знаходилися в **st2**,наприклад:

**char st1 [ ] = “Borland С++5”;**

**char st2 [ ] = ” narlBod “;**

**int k;**

**k= strspn (sti, st2);** — змінна **k**одержує значення, що дорівнює **8**, тому що перші 8 символів рядка містилися в **st1**(враховуючи символ пропуску);

* **char \*strstr (const char \*st1, const char \*st2);** — функція шукає в рядку **st1** перше входження **st2** і повертає покажчик на перший символ, знайдений у **st1**, з підрядка**st2**; якщо ря­док **st2** не виявлений в **st1**, функція повертає **0**, наприклад:

**char stl [ ] = “Привет, сокурсник, идем на экзамен”;**

**char st2[ ] = “сокурсник”;**

**char spt;**

**spt = strstr (stl, st2);**

Результат виконання:

**spt = “сокурсник, идем на экзамен”.**

За потреби визначення останнього входження можна спо­чатку реверсувати рядок за допомогою функції **strrew;**

* **char \*strtok (char \*st, const char \*dlm);** — розбиття рядка на лексеми (сегменти), обмежені символами, що входять до складу рядка **dim**. Цей параметр може містити будь-яку кіль­кість різних обмежників — ознак границь лексем, після виділення лексеми в рядок **st** поміщається символ **«\0»**.

Наступні виклики функції **strtok()** повинні бути з першим аргументом **NULL.** Бони будуть повертати покажчик на інші, наявні в **st** лексеми. Щоразу після завершення виділення лексеми у її кінці замість розділового символу поміщається сим­вол **«\0**». Після того, як у рядку не залишиться жодної лексеми, функція повертає **NULL**. Для збереження вихідного рядка його треба записати в резервну змінну. Цю функцію зручно використовувати для розбиття речення на слова або будь-які інші сегменти. Розглянемо приклад програми з використан­ням функції **strtok().**

***Приклад 1.*** Скласти програму, яка вводить речення, здійснює розбиття його на слова, підраховує кількість символів у кожному сло­ві та виводить відповідну інформацію

// СРР — применение функции strtok()

/\* определение порядкового номера слова в предложении и подсчет количества символов в каждом слове \*/

**#include <string.h>**

**#include <iostream.h>**

**#include <conio.h>**

**voidmain (void)**

**{ char \*tk, \*spt=", .!";**

**char st[ ] = "Делай великое, не обещая великого.";**

**cout << st<< endl;**

**int і = 1;**

**tk = strtok (st, spt);**

**while (tk != NULL)**

**{**

**cout << і << " слово — " << tk << " — содержит " << strlen(tk) << " символов" << endl;**

**tk = strtok(NULL, spt); і++;}**

**getch ();**   // задержка экрана

**}**

Результати виконання програми:  
**Делай великое, не обещая великого.**  
**1  слово — Делай — содержит 5 символов**  
**2  слово — великое — содержит 7 символов**  
**3   слово — не — содержит 2 символов**

**4   слово — обещая — содержит 6 символов**  
**5   слово — великого — содержит 8 символов**  
Процес розбиття речення на слова можна було б здійснити з використанням і такого програмного фрагмента:  
**tk = strtok (st < spt);** *// первый вызов функции*  
**while (tk)**  
**if ((tk = strtok(), spt) != 0) cout <<. . ..**  
***Для видалення з рядка підрядка*** або ***символа із заданої позиції*** у бібліотеці **string.h**немає спеціальної функції, однак можна написати власну, наприклад:

**void del (char \*st, int k, int n);**  
**{**  
**for (int і = k; і < strlen(st); i++)**  
**st[i] = st [i + n];**  
**st[i] =’\0′;**               *// запись “\0” в конец новой строки*  
**}**  
де **st** — вихідний рядок (покажчик на нього);  
**n** — кількість символів у підрядку, що вилучається;  
**k** — позиція, з якої треба вилучити підрядок. Наведемо приклад, котрий ілюструє використання функції **void del ();** .

***Приклад 2.*** Скласти програму вилучення підрядка в **n** символів з **k**-ої позиції в рядку.

/\* *2.СРР — удаление подстроки в п символов из k-ой позиции в строке* \*/

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

*//--------------- функция удаления подстроки из строки*

**void del (char\*sp, int k, int n)**

**{ int і;**

**for (і = k; і < strlen(sp); і++)**

**sp[i] = sp[i+n];**

**sp[i] = \0';}**

**main ()**           *//----------- главная функция*

**{ char st[50], pst[10];**

**cout << "\*\*\*\*\* Введите строку\n";**

**cin.getline(st, 50);**

**cout << "\*\*\*\*\* Введите подстроку\n";**

**сіn >> pst;**

**cout << "Исходная строка: — "<< st << endl;**

**del (st, strstr(st, pst)-st, strlen(pst));**

**cout << "Новая строка: — "<<st<<endl;**

**getch();**

**}**

Результата обчислень:  
**\*\*\*\*\* Введите строку**  
**Люблю писать программы на языке С++!**  
**\*\*\*\*\* Введите подстроку**  
**писать**  
**Исходная строка: — Люблю писать программы на языке С++!**  
**Новая строка: — Люблю программы на языке С++!**

* **void\* memchr (const void \*st, int s, int n);** — функція шукає символ **“s“** у рядку **\*st**довжиною **n** байт, тобто в блоці пам’яті, на який вказує покажчик **st**. Якщо символ знайдений, функція повертає покажчик на цей символ, а в протилежному випадку — повертає **NULL**;

Приклад.

#include <stdio.h>

#include <string.h>

int main(void)

{

char \*p;

p = memchr("строка из примера", ' ', 17);

printf(p);

return 0;

}

Щоб не розраховувати кількість символів пропонується використовувати функцію **strlen.** Викликфункції знижує ефективність, цього можна уникнути використовуючи операцію **sizeof (str) /sizeof (char)**

* **void\* memcmp (const void \*s1, const void \*s2, n);** і **void\* memicmp (const void \*sl,const void \*s2, int count);** — функції порівнюють **n** байт з двох буферів, на початок яких указують **s1 і s2**.

Функції повертають значення

**<0 — якщо s1 < s2;**  
**=0 — якщо s1 = s2;**  
**>0 — якщо s1 > s2;;**

* **char \*strset (char \*st, int ch, int n);** і **char \*strset (char \*st, int ch);** — функції заповнюють рядок **st** символом **ch** і повертають покажчик на отриманий рядок, **n** —заповнює **n** сим­волів рядка **st.**

**Функції перетворення рядків у числа та чисел у рядки знаходяться у файлі stdlib.h:**

* **int atoi (const char \*s);** — перетворює рядок **s** у число типу **int**. Повертає отримане число **0**, якщо зустрінеться символ, що не може бути перетворений. Рядок повинен містити число, наприклад, **«2345»**, та мати таку структуру: **[пропуски] [знак числа] [цифри];**
* **long atol (const char \*s);** — перетворює рядок **s** у число типу **long** **int** **(**аналогічна функції **atoi.**);
* **double atof (const char \*s);** — перетворює рядок символів у число з плаваючою крапкою типу **double**. Якщо зустрічається символ, що не може бути перетворений, повертає **0**. Оброблюваний рядок повинен мати таку структуру: **[пропуски] [знак числа] [цифра.цифра] [літера е, Е, d або D] [знак порядку] [цифри порядку]**, наприклад, **«-12345.123» або «-12.345123 ЕЗ»**;
* **char \*ecvt (double vl, int n, int \*dec, int \*sign);** — перетворює число **vl** у рядок символів, кількість яких дорівнює **n** символів цифр. Положення десяткової крапки від першої цифри числа повертається до змінної, на яку вказує **dec**. Знак числа повертається до змінної, на яку вказує **sign**. Якщо **sign = 0**, то число є додатним, інакше — від’ємним. Отриманий рядок збе­рігається у внутрішній пам’яті функції, покажчик повертається на початок сформованого рядка;
* **char \*fcvt (double vl, int n, int \*dec, int \*sign);** — аналогічна до попередньої функції **char \*ecvt()**, але якщо для функ­ції **ecvt** параметр **dec** задає загальну кількість цифр, то для функції **fcvt** — кількість цифр після десяткової крапки;
* **char \*gcvt (double vl, int n, char \*buf);** — перетворює число **vl** у рядок, котрий поміщає в буфер, покажчик на по­чаток якого є **buf**, **n** — число цифр у символічному записі пе­ретвореного числа. Отриманий рядок містить символ знака числа і десяткової крапки, якщо число містить менше десяткових цифр, ніж **n**. У цьому випадку молодша цифра дробової час­тини відкидається. Якщо перетворене число не можна поміс­ити в задану кількість цифр **n**, функція генерує символьний запис в експоненціальній формі із символом **Е** і знаком порядку. Функція повертає покажчик на початок сформованого рядка;
* **strlen (st)** — повертає довжину змінної **st** без нуль-термінала **«\0»**.

***Функції перевірки символів знаходяться у файлі ctype.h:***

* **isgraph (s)** — повертає значення «істина» (**1**), якщо s є друкованим символом, і «неправда» (**0**), якщо **s** є пропуском або яким-небудь не відображуваним символом;
* **isprint (s)** — повертає значення «істина» (**1**), якщо s є дру­кованим символом, включаючи символ пропуску, і «неправда» (**0**) у всіх інших випадках;
* **ispunct (s)** — повертає значення «істина» (**1**), якщо **s** є зна­ком пунктуації (будь-який друкований символ, крім пропуску), і «неправда» (**0**) в інших випадках;
* **isdigit (s)** — повертає значення «істина» (**1**), якщо s є циф­рою від 0 до 9, і «неправда» (**0**) в інших випадках;
* **isalmim (s)** — повертає значення «істина» (**1**) якщо s є циф­рою або літерою (заголовною або строковою), і «неправда» (**0**) у всіх інших випадках (тобто перевіряє алфавітні та цифрові символи).

***Функції перетворення символів:***

* **tolower (s)** — перетворює символ s до нижнього регістра;
* **toupper (s)** — перетворює символ s до верхнього регістра;
* **atoi (s)** — перетворює рядок s до цілого числа;
* **atol (s)** — перетворює рядок s до довгого цілого;
* **atof (s)** — перетворює рядок s до числа з плаваючою крапкою.

Розглянемо приклади з використанням рядкових функцій.  
  
***Приклад 3.*** Ввести до пам’яті комп’ютера список прізвищ, які розташовані в будь-якому порядку, та відсортувати їх за алфавітом.  
Розглянемо перший варіант (***див. 3\_1.СРР***) реалізації поставленої задачі. Будемо вважати, що вводять прізвища та ініціали, тоді програма може мати вигляд:

// *3 1.СРР — отсортировать фамилии по алфавиту*

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**main( )**

**{ const int n=5;**

**char sp[n][l5], r[15];**

**int i, k;**

*//------------------------------- ввод фамилий и инициалов*

**cout<< "\*\*\*\*\* Введите " << n << " фамилий \n";**

**for (і = 0; і < n; і++)**

**{ cout<<"Введите "<<(і+1)<<" фамилию и инициалы\n";**

**cin.getline (sp[i], sizeof (sp[i]) - 1);**

**}**

*//------------------------------ сортировка списка фамилий*

**for (k = 1; k < n; k++)**

**for (i = 0; і < n-k; i++)**

**if (strcmp (sp[i], sp[i+l])>0)**

**{ strcpy (r, sp[i]);**

**strcpy (sp[i], sp[i+1]);**

**strcpy (sp[i+1], r);}**

**cout<<"\n Отсортированный массив фамилий \n";**

**for (і = 0; і < n; i++)**

**cout << sp[i] << endl;**

**getch ();**

**}**

Результати обчислень:  
**\*\*\*\*\*\*\* Введите 5 фамилий**  
**Введите 1 фамилию и инициалы**  
**Иванченко С. И.**  
**Введите 2 фамилию и инициалы**  
**Авдиенко А. Р.**  
**Введите 3 фамилию и инициалы**  
**Яшин Б. Ю.**  
**Введите 4 фамилию и инициалы**  
**Кашкин Т. Б.**  
**Введите 5 фамилию и инициалы**  
**Мельниченко Т. Ю.**  
**Отсортированный массив фамилий**  
**Авдиенко А. Р.**  
**Иванченко С. И.**  
**Кашкин Т. Б.**  
**Мельниченко Т. Ю.**  
**Яшин Б. Ю.**

У наведеній програмі використано масив прізвищ **sp [6][15]** і символьний рядок **r**, який потрібен для тимчасового зберігання прізвища при сортуванні масиву. Для сортування був метод виштовхування («пухирця»).

Порівняння елементів символьного масиву **(char sp[n][15])** здійснюється за допомогою функції **strcmp( )**, а перезапис прізвищ з одного елемента масиву **sp[i]** в другий — **sp[i+1]**— за допомогою функції **strcpy( )** і змінної **r**.

Після сортування на екран виведено одержаний масив. Другий варіант (***див. 3\_2.СРР***) розв’язання поставленої задачі використовує покажчики.

/\*  *3\_2.СРР — сортировка списка фамилий в алфавитном порядке с использованием указателей* \*/

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**void main( )**

**{ const n=5;**

**char sp [n][15];**

**int i, k;**

**char \*ps[n], \*ptr;** //*ps[n] — массив указателей*

*// ввод фамилий и инициализация массива указателей*

**cout << "\*\*\*\*\* Введите фамилии \n";**

**for (і = 0; і < n; і++)**

**{ gets (sp [і]);**

**ps[i] = sp[i];}**

*//--------------------------- вывод исходной информации*

**cout << "\n\*\*\*\*\* Исходный список\n";**

**for (і = 0; і < n; і++)**

**puts (ps[i]);**

//------------------------------------ сортировка массива

**for (k = 1; k < n; k++)**

**for (i = 0; i<n-k; i++)**

**if (strcmp (ps[i], ps[i+1]) > 0)**

**{ ptr = ps[i];**

**ps[i] = ps[i+1];**

**ps[i+1] = ptr;   }**

*//--------------------- вывод отсортированного массива*

**cout << "\n\n\*\*\*\*\*Отсортированный список \n";**

**for (і = 0; і < n; і++)**

**puts (ps[i]);**

**getch ();**

**}**

Результати виконання програми:  
**\*\*\*\*\* Введите фамилии**  
**Игнатенко А. P.**  
**Головко Н. А.**  
**Долбня Б. В.**  
**Андриенко С. Ф.**  
**Ичко Т. В.**  
**\*\*\*\*\* Исходный список**  
**Игнатеяко А. Р.**  
**Головко Н. А.**  
**Долбня Е. В.**  
**Андриенко С. Ф.**  
**Ичко Т. В.**  
**\*\*\*\*\* Отсортированный список**  
**Андриенко С. Ф.**  
**Головко Н. А.**  
**Долбня Е. В.**  
**Игнатенко А. Р.**  
**Ичко Т. В.**  
  
***Приклад 4.*** Ввести рядок і видалити в ньому зайві пропуски.

//  *4.СРР — удаление лишних пробелов в строке*

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**void main()**

**{ char st[ ] = "Краткость — сестра таланта";**

**int і, n = 0;**    *//n — для подсчета пропусков*

**for (і = 0; і < strlen(st); i++)**

**{ if (st[i] != ' ')**

**{ cout « st[i];**

**n=0; }**

**else n++;**

**if (n == 1)**

**cout << st[i];   }**

**getch();**

**}**

Результати виконання програми:  
**Краткость — сестра таланта**  
***Приклад 5.*** Визначити позицію входження підрядка в рядок.

// *5.СРР — поиск позиции вхождения подстроки в строку*

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**const int m = 50;**

**void main()**

**{ char \*pt, mainstr[m], substr[m];**

**int n, k = 0;**

**cout << "\*\*\*\*\* Введите строку " << endl;**

**cin.getline (mainstr, m);**

**cout << "\*\*\*\*\* Введите подстроку " << endl;**

**cin.getline (substr, m);**

**pt = strstr (mainstr, substr);**

**cout << endl;**

**while (pt)**

**{ k++;**      *// номер вхождения*

**n = pt - mainstr;**

**cout << k <<" -oe вхождение подстроки" <<" номер позиции = "<< n << endl;**

**pt = strstr(++pt, substr);**

**cout << k << " " << \*pt << endl;}**

**if (k == 0) cout <<"Подстрока не содержится в строке"<<endl;**

**getch();**

**}**

Результат виконання:  
**\*\*\*\*\* Введите строку**  
**Почти во всех делах самое трудное — начало.**  
**\*\*\*\*\* Введите подстроку**  
**самое трудное**  
**1-ое вхождение подстроки номер позиции = 20**  
***Приклад 6.*** Знайти заданий символ у рядку.

// *6.СРР — поиск символа в строке*

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**main( )**

**{ const int m = 50;**

**char sim, \*pt, str[m];**

**int n, k = 0;**

**cout << "\*\*\*\*\* Введите строку" << endl;**

**cin.get (str, m);**

**cout << "\*\*\*\*\* Введите символ" << endl;**

**сіn >> sim;**

**pt = strchr (str, sim);**

**while (pt)**

**{ k++;**

  **n = pt-str;**

**cout<<k<<"-я позиция вхождения символа = "<<n<<endl;**

**pt = strchr(++pt, sim); }**

**cout << "Количество вхождений = "<<k<<endl;**

**if (k==0) cout << "Символ не входит в строку" << endl;**

**getch();**

**}**

Результати обчислень:  
**\*\*\*\*\* Введите строку**  
**Алгоритм — фундаментальное понятие информатики**  
**\*\*\*\*\* Введите символ**  
**и**  
**1-я     позиция входения символа = 5**  
**2-я     позиция входения символа = 32**  
**3-я     позиция входения символа = 35**  
**4-я     позиция входения символа = 43**  
**5-я     позиция входения символа = 45**  
**Количество вхождений = 5**

***Приклад 7.*** З уведеного списку прізвищ (без ініціалів) вилучити такі, що починаються на задану літеру і мають задане закінчення, та вивести повідомлення про прізвище з найменшою кількістю літер.

/\* *7.CPP — найти фамилии, начинающиеся на заданную букву и имеющие заданноеокончание* \*/

**#include <iostream.h>**

**#include <string.h>**

**#include <conio.h>**

**main()**

**{ const int n = 6;**

**char spis [n][15], pok[5], p;**

**int i, minfam, k=0;**

*//------------------------------ ввод списка фамилий без инициалов*

**cout << "\*\*\*\*\* Введите "<<n<<" фамилий\n";**

**for (і = 0; і < n; i++)**

**сіn >> spis [і];**

**cout << "\*\*\*\*\* Введите первую букву\n";**

**сіn >> р;**

**cout << "\*\*\*\* Введите окончание\n";**

**сіn >> роk;**

// определение фамилии на заданную букву и на заданное окончание

**cout << "\*\*\*\*\* Искомые фамилии\n";**

**for (і = 0; і < n; і++)**

**if (spis [i][0]==p && strcmp(strrchr(spis[i], pok[0]), pok)==0)**

**cout<<spis [i]<<endl;**

*//-------------- поиск фамилии с наименьшим количеством букв*

**minfam - strlen (spis [0]);**

**for (і = 1; і < n; і++)**

**if (strlen (spis [і]) < minfam)**

**{ minfam = strlen (spis[i]);**

**k=і; }**

**соut<<"\Фамилия с наименьшим колич. букв — "<<spis [k]<<" \n";**

**cout << "Её длина = " << strle(spis [k]) << " символов\n";**

**getch ();**

**}**

Результати обчислень:  
**\*\*\*\*\* Введите 6 фамилий**  
**Андриенко**  
**Коваленко**  
**Степаненко**  
**Курко**  
**Коноваленко**  
**Перекотиполе**  
**\*\*\*\*\* Введите первую букву**  
**К**  
**\*\*\*\*\* Введите окончание**  
**ко**  
**\*\*\*\*\* Искомые фамилии**  
**Коваленко**  
**Курко**  
**Коноваленко**  
**Фамилия с наименьшим колич. букв — Курко**  
**Её длина = 5 символов**

У програмі для зберігання першої літери використовується змінна з ім’ям **р**, а для закінчення — змінна **роk**. Вилучення закінчення прізвища здійснює функція **strrchr(spis[i],pok[0])**, яка повертає покажчик на останнє входження заданої літери **(рок[0])** у рядок**(spis[i])**. Потім цей покажчик порівнюється з введеним закінченням прізвища за допомогою функції **strcmp()**. У цілому в операторі **if**… визначається як перша літера, так і закінчення прізвища. Змінна **minfam** існує для знаходження прізвища з най­меншою кількістю літер згідно з алгоритмом визначення мінімального елемента масиву (див. ***приклад 1.3***).

***Контрольні запитання для самоперевірки***.

1. Що таке рядки та значення елементів символьного типу?
2. Що являє собою масив символьного типу?
3. Як здійснюється введення символьних даних?
4. Як виконується порівняння даних символьного типу?
5. Наведіть приклад використання операції конкатенації.
6. Як визначити кількість символів у рядку?
7. Які функції мови С++ необхідні для виділення підрядка з рядка?
8. Які функції здійснюють перевірку символів?
9. Як виконується перетворення рядків у числа і навпаки?
10. Охарактеризуйте функції пошуку підрядка в рядку.
11. Які функції здійснюють перевірку символів?
12. Як можна здійснити видалення підрядка з рядка або символу із заданої позиції?

**Для самостійного вивчення** *(2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

**Рекомендована література**

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Павловская Т.А. С/С++. Программирование на языке высокого уровня. СПб.: Питер, 2003. – 461 с. URL: <http://www.ph4s.ru/bookprogramir_1.html>
3. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с. URL: <http://ijevanlib.ysu.am/wp-content/uploads/2018/03/deytel.pdf>