**Лабораторна робота №8.Написання програм з використанням символьних та рядкових величин**

**Мета**: навчитися писати програми з використанням символьних та рядкових величин, структур та масивів **на мові С++** в **консольному режимі.**

**Методичні вказівки щодо організації самостійної роботи студентів**

1. Прочитати (повторити) основні теоретичні відомості. Переглянути лекцію №11 "Покажчики, символьні та рядкові величини ", а також розгляд прикладів програм з символьними та рядковими величинами в практичній роботі №8. "Розгляд прикладів програм з символьними та рядковими величинами, масивами та структурами".
2. Запустити середовище програмування С++ .
3. Записати програму, що виконує 3 завдання з пп.4.1-4.3. В першому рядку кожної програми записати

*// Група № Прізвище*

вказавши номер своєї групи та своє прізвище.

Як і в попередніх ЛР, програма повинна запитати номер завдання (число 1, 2, 3 або 0 для закінчення) і в залежності від введеного значення виконувати відповідне завдання. Якщо введений 0 – програма припиняє роботу.

Вхідні дані ввести, а результати вивести, використовуючи потокове введення-виведення даних.

1. Завдання:
   1. Написати програму, яка питає ім`я, порівнює з тими, що вона має як елементи символьного масиву (рядки типу string) та вітає або повідомляє, що “не знайома”.

Елементи символьного масиву задати через ініціалізацію (див. приклад в кінці теоретичних відомостей).

* 1. В заданому рядку "Don’t cut the bough you are sitting on." знайти і вивести найкоротше і найдовше слово, вказати номери позицій, з яких вони починаються. ***Вказівки:*** перегляньте потрібні функції (пошук символу, виділення підрядка, знаходження довжини рядкової змінної). Вам може буди потрібно ввести проміжні рядкові зміні, а для організації перегляду в циклі булеву змінну. Продумайте алгоритм.
  2. Розробити програму, яка запитує текстовий рядок, і в текстовому рядку замінює будь-яку кількість однакових символів, що йдуть один за одним підряд на один такий же символ та цифру, яка відповідає кількості видалених символів (Наприклад: ‘*1CABk3KKK111DeFf0100fk0cccccc*’=*’1CABk3K212DeFf0101fk0с5*’). ***Вказівки:*** початковий рядок вводити з клавіатури; отриманий рядок виводити під початковим рядком.

**Основні теоретичні відомості**

Рядок – це послідовність символів, яка закінчується нульовим байтом *‘\0’*. Дана послідовність символів записується в подвійних лапках.

Послідовності в С++ представляються масивами або покажчиками, тому стандартно програми мовою C++ зберігають рядки як масиви типу *char*. Для оголошення строкової змінної необхідно всередині програми оголосити масив типу *char* з кількістю елементів, достатньою для зберігання необхідного рядка символів.

Головна відмінність між символьними рядками та іншими типами масивів полягає в тому, що C++ визначає останній елемент масиву. Програми мовою C++ представляють кінець символьного рядка за допомогою символу NULL, який в C++ зображується як спеціальний символ '\0', або нульовий байт.

Створення рядків, наприклад, відбувається наступним чином:

char str2[10]="Hello";

Для стандартної роботи з символьними рядками в стилі С++ призначена бібліотека string.h, яка містить наступні основні функції:

– **char \**strcat*(char \*dest, const char \*src);** – дописує рядок *src* в кінець *dest*;

– **char \**strncat*(char \*dest, const char \*src, size\_t n);** – дописує не більше *n* початкових символів рядка *src* (або весь *src*, якщо його довжина менше) у кінець *dest*;

– **char \**strchr*(const char \*str, int ch);** – шукає символ *ch* у рядку *str*, починаючи з голови, і повертає його адресу, або NULL якщо символ не знайдено;

– **char \**strrchr*(const char \*str, int ch);** – шукає символ *ch* у рядку *str*, починаючи з хвоста і повертає його адресу, або NULL якщо символ не знайдено;

– **int *strcmp*(const char \*str1, const char \*str2);** – лексикографічне порівняння рядків;

– **int *strncmp*(const char \*str1, const char \*str2, size\_t n);** – лексикографічне порівняння перших *n* байтів рядків;

**– char \**strcpy*(char \*dest, const char \*src);** – копіює рядок з *src* у *dest*;

– **char \**strncpy*(char \*dest, const char \*src, size\_t n);** – копіює до *n* байт рядку з *src* у *dest*;

– **size\_t *strlen*(const char \*str);** – повертає довжину рядка;

– **size\_t *strspn*(const char \*str, const char \*alp);** – визначає максимальну довжину початкового підрядка з *str*, що складається виключно з байтів, перерахованих в *alp*;

– **char \**strpbrk*(const char \*str, const char \*alp);** – знаходить перше входження будь-якого символу, перерахованого в *alp*, у *str*;

– **char \**strstr*(const char \*str, const char \*pstr);** – знаходить перше входження рядка *pstr* у *str*.

Однак окрім розглянутого вище стандартного способу роботи з символьними рядками в бібліотеці STL С++ є ще одна бібліотека для роботи з рядками. Вона дозволяє легко вирішувати наступні задачі з високою ефективністю:

– створювати, присвоювати, копіювати і видаляти рядки;

– виконувати перетворення типів символьних змінних;

– порівнювати рядки;

– поєднувати рядки;

– визначати довжину рядка;

– знаходити і заміщати потрібний фрагмент у рядку.

Для використання даної бібліотеки необхідно підключити заголовний файл <string>. Після цього операція створення нового рядка виявиться настільки ж простою, як і створення змінної будь-якого базового типу.

**string** hi(“hello”);

**string** lo=“greetings”;

**string** es=””;

Рядковим змінним можна присвоювати значення, як і змінним будь-яких інших типів:

**string** name(“Fred”);

name = “Paul”;

При цьому всі операції розподілу пам'яті будуть виконані коректно.

Операції + (конкатенація), операція порівняння (==, > тощо) дозволяють легко виконувати відповідні дії над рядками.

**Приклад програми з використанням масиву рядкових величин.**

**#include <iostream>**

**#include <string>**

**#include <Windows.h>**

**using namespace std;**

**//-------------------------------------------------------**

**int main()**

**{**

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

**string month[5]={"april","may","june","july","august"};**

**string name;**

**bool found=false;**

**cout << "What month? ";**

**cin >> name;**

**for (int i=0;i<5;i++)**

**if (name==month[i])**

**{cout << "Hello, " << name << "!\n";**

**found=true;}**

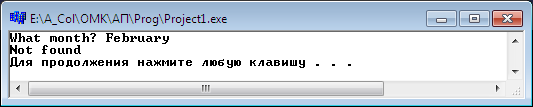
**if (!found ) cout << "Not found " <<endl;**

**system("pause");**

**return 0;**

**}**

**Результат роботи програми**



**Деякі зауваження щодо роботи з символьним типом даних char**

***Тип даних char***

Змінна типу char займає 1 байт. Однак, замість конвертації значення типу char в ціле число, воно інтерпретується як ASCII-символ.

**ASCII** (від англ. “**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange”) — це американський стандартний код для обміну інформацією, який визначає спосіб представлення символів англійської мови (+ декілька інших) у вигляді чисел від 0 до 127. Наприклад: код букви 'а' — 97, код букви 'b' — 98. Символи завжди поміщаються в одинарні лапки.

Символи від 0 до 31 в основному використовуються для форматування виводу. Символи від 32 до 127 використовуються для виведення. Це літери, цифри, знаки пунктуації, які більшість комп’ютерів використовує для відображення тексту (англійською мовою).

Наступні два оператори виконують однакові дії - присвоюють змінним типу char ціле число 97:

**char ch1(97);** // ініціалізація змінної типу char цілим числом 97

**char ch2('a');** // ініціалізація змінної типу char символом 'a' (97)

При використанні фактичних чисел для представлення символів (з таблиці ASCII) потрібна уважність. Наступні два оператори виконують різні дії:

**char ch(5);** // ініціалізація змінної типу char цілим числом 5

**char ch('5');** // ініціалізація змінної типу char символом '5' (53)

***Оператор static\_cast***

Якщо ви хочете вивести символи у вигляді цифр, а не у вигляді букв, то потрібно повідомити cout виводити змінні типу char в вигляді цілочисельних значень. Можна присвоїти змінній типу int змінну типу char і вивести її, але це не дуже хороший спосіб:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**char ch(97);**

**int i(ch); // присвоюємо значенням змінної ch змінній типу int**

**cout << i << endl; // виводимо значення змінної типу int**

**return 0;**

**}**

Результат:

97

Кращим способом є конвертація змінної з одного типу даних в інший за допомогою **оператора static\_cast**.

**Синтаксис static\_cast виглядає наступним чином:**

static\_cast<новий\_тип\_даних>(вираз)

static\_cast приймає значення з (вираз) в якості вхідних даних і конвертує його у вказаний вами <новий\_тип\_даних>.

Приклад використання оператора static\_cast для конвертації типу char в тип int:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**char ch(97);**

**cout << ch << endl;**

**cout << static\_cast<int>(ch) << endl;**

**cout << ch << endl;**

**return 0;**

**}**

Результат виконання програми:

a  
97  
a

**static\_cast** приймає (вираз) в якості вхідних даних. Якщо ми використовуємо змінну (в виразі), то ця змінна змінює свій тип тільки в інструкції з оператором static\_cast. Процес конвертації ніяк не впливає на вихідну змінну з її значенням! В наведеному вище прикладі, змінна ch залишається змінною типу char з колишнім значенням, чому є підтвердженням останній оператор з cout.

В static\_cast немає ніякої перевірки діапазону, тому при використанні занадто великих або занадто маленьких чисел для конвертованого типу, то відбудеться **переповнення**.

***Введення символів***

Наступна програма просить користувача ввести символ. Потім вона виводить цей символ і його ASCII-код:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**cout << "Input a keyboard character: ";**

**char ch;**

**cin >> ch;**

**cout << ch << " has ASCII code " << static\_cast<int>(ch) << endl;**

**return 0;**

**}**

Результат виконання програми:

Input a keyboard character: q  
q has ASCII code 113

Зверніть увагу, що навіть якщо cin дозволить вам ввести декілька символів, змінна ch буде зберігати тільки перший символ (саме він і розміщується у змінній). Інша частина користувацького введення залишиться у вхідному буфері, який використовує cin і буде доступна для використання наступним викликам cin.

Приклад:

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**cout << "Input a keyboard character: ";**

**// припустимо, що користувач ввів "abcd"**

**char ch;**

**cin >> ch;**

**// ch = 'a', "bcd" залишається у вхідному буфері**

**cout << ch << " has ASCII code " << static\_cast<int>(ch) << endl;**

**// Звернуть увагу, що наступний cin не просить**

**// користувача що-небудь ввести, дані підтягуються з**

**// вхідного буферу!**

**cin >> ch; // ch = 'b', "cd" залишається в буфері**

**cout << ch << " has ASCII code " << static\_cast<int>(ch) << endl;**

**return 0;**

**}**

Результат виконання програми:

Input a keyboard character: abcd  
a has ASCII code 97  
b has ASCII code 98

***Розмір, діапазон і знак типу сhar***

В С++ для змінних типу char завжди виділяється 1 байт. За замовчуванням, char може бути як **signed, так і unsigned** (хоча зазвичай signed). Якщо ви використовуєте char для зберігання ASCII-символів, то вам не потрібно вказувати знак змінної (так як і signed, і unsigned можуть містити значення від 0 до 127).

Але якщо ви використовуєте тип char для зберігання невеликих цілих чисел, то тоді слід уточнити знак. Змінна типу char signed може зберігати числа від -128 до 127. Змінна типу char unsigned має діапазон від 0 до 255.

***Що використовувати: ‘\n’ чи endl (std::endl)?***

При використанні cout, дані для виведення можуть поміщатися в буфер, тобто cout може не відправляти дані відразу ж на виведення. Замість цього він може залишити їх при собі на деякий час. Це робиться в цілях підвищення продуктивності.

І '\n', і endl обидва переносять курсор на наступний рядок. Тільки endl ще гарантує, що всі дані з буферу будуть виведені, перед тим, як продовжити.

Коли використовувати '\n', а коли endl?

Використовуйте endl, коли потрібно, щоб ваші дані виводилися відразу ж (наприклад, під час запису в файл або при оновленні індикатора стану будь-якого процесу). Зверніть увагу, це може спричинити за собою незначне зниження продуктивності, особливо якщо запис на пристрій відбувається повільно (наприклад, запис файлу на диск).

Використовуйте '\n' у всіх інших випадках.

***Різниця між одинарними і подвійними лапками при використанні з символами***

Символи завжди поміщаються в одинарні лапки (наприклад, 'а', '+' чи '5'). Змінна типу char представляє тільки один символ (наприклад, літеру а, символ + чи число 5). Некоректним записом буде:

**char ch('56'); // змінна типу char може містити тільки 1 символ**

Текст, який знаходиться в подвійних лапках, називається рядком (наприклад, "Hello, world!"). **Рядок** (**тип string**) — це набір послідовних символів.

Ви можете використовувати літерали типу string в коді:

**cout << "Hello, world!";**

**// "Hello, world!" - це літерал типу string**

**Покажчики**

**Покажчики** — це змінні, котрі містять адресу пам’яті, розподіленої для об’єкта відповідного типу. При оголошенні змінної-покажчика слід вказати тип даних, адресу яких буде містити змінна, та ім’я покажчика з символом «\*».

Загальний формат опису покажчика має вигляд:

**тип \*ім’я;**

де **тип** — тип значень, на який вказує покажчик;  
**ім’я** — ім’я змінної-покажчика;  
«\*» — операція над типом, що читається «покажчик на тип».

Наприклад:

**int \*рn** – покажчик на ціле значення;  
**float \*pf1, \*pf2;** — два покажчики на дійсні значення.

Покажчики не прив’язують дані до якого-небудь визначеного імені змінної і можуть містити адреси будь-якого неіменованого значення. Існує адресна константа **NULL**, що означає порожню адресу.

Мова C++ налічує лише дві операції, які стосуються адрес змінних, а саме:

**«&»** — **операція взяття адреси** («адреса значення»);

**«\*»** — **операція розіменування** («значення за адресою»).

Операція взяття адреси **«&»** застосовується разом зі змінною і повертає адресу цієї змінної. Операція розіменування «\*» використовується разом з покажчиками і бере значення, на яке вказує змінна-покажчик, розташована безпосередньо після символу «\*».

Оголошення покажчиків можна здійснити одним з таких способів:

**<тип> \*ptr;**  
**<тип> \*ptr = <змінна-покажчик>;**  
**<тип> \*ptr = &<ім’я змінної>;.**

Наприклад:  
**int \*ptx, b; float у;** — оголошені змінна-покажчик **ptx** та змінні **b і у**;

**float \*sp = &у;** — покажчику **sp** присвоюється адреса змінної **у**;

**float \*р = sp;** — покажчику **р** присвоюється значення (адреса значення), яке міститься в змінній **sp**, тобто адреса змінної **у**.

При оголошенні покажчиків символ «\*» може знаходитися перед ім’ям покажчика або відразу після оголошення типу покажчика і поширювати свою дію тільки на одну змінну-покажчик, перед якою він записаний:

**long \*pt;   long\*Uk;   int \*ki, x, h;** — оголошення описів.

За потреби для опису покажчика на комірку довільного типу замість ідентифікатора типу записується слово **void**, а саме:

**void \*р, \*pt;** — опис двох покажчиків на довільний тип даних.

**Перед використанням покажчика у програмі його обов’язково необхідно ініціювати**, іншими словами, необхідно присвоїти адресу якого-небудь даного, інакше можуть бути непередбачені результати.

Для одержання доступу до значення змінної, адреса якої зберігається в покажчику, досить у відповідному операторі програми записати ім’я покажчика з символом «\*» — здійснити операцію розіменування.

Розглянемо фрагмент програми з поясненнями:

**int \*р, \*р1;** — оголошені два покажчики на комірку пам’яті типу **int**;

**int х = 12, у = 5, m[7];** — оголошені змінні **х**, **у** і масив **m**, змінні ініційовані;

**р = &у;**     // р (&у); — покажчику **р** присвоєна адреса змінної **у**.

**cout << “Адреса р ” << р << “Значення за цією адресою = ” << \*р; ,**

Виведеться адреса комірки пам’яті, де записана змінна **у** і значення цієї змінної (**тобто 5**).

Використовуючи запис **х = \*р;**, одержимо **х = 5** тому, що **\*р = у = 5;**.

Змінити величину параметра **у** можна так:

**у = 10;**         // \*Р= 10;  
**\*р = \*р+5;**    //у +=5;.

Остання операція означає збільшення значення змінної цілого типу на **5**, тобто **у= 15**.

При ініціюванні покажчиків їм можна присвоювати або адресу об’єкта (змінної), або адресу конкретного місця пам’яті (масиву), або число 0 (нуль), а саме:

**int \*pt = (char \*) 0x00147;** — присвоюється адреса комірки;

**int \*arrpt = new int [10];** — визначається початкова адреса розміщення динамічного масиву;

**char \*р = 0;** — здійснюється ініціювання нулем.

Оскільки покажчики — це спеціальні змінні, то в операціях з іншими покажчиками вони можуть використовуватися без символу «\*», тобто без розкриття посилання, наприклад:

**float \*pt1, \*pt2, х=15, m[5];**

**pt1 = &x;**

**pt2 = pt1;**

**pt1 = m**;         //pt1 = &m[0];

де **m** — ім’я масиву, що розглядається як спеціальний покажчик-константа.

**Приклад**

**#include <iostream.h>**

**#include <Windows.h>**

**using namespace std;**

**int main ( )**

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

**int x = 10;**

**int \*px (&x); // int \*px = &x;**

**cout << "x =" << x << endl;**

**cout << "\*px =" << \*px << endl;**

**x \*= 2; //x=x\*2;**

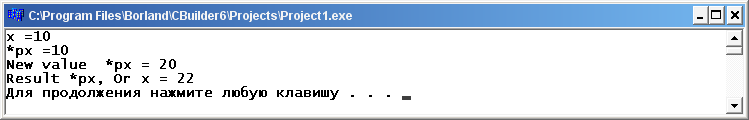
**cout << "New value \*px = " << \*px << endl;\***

**px += 2; // \*px=\*px + 2;**

**cout << "Result \*px, Or x = " << x << endl;**

**system("pause"); //затримка екрану**

**}**

Результат виконання програми:  


Для змінної-покажчика існує своя адреса і тому будуть доцільними записи:

**int \*pt1, \*pt2;**

**pt1 = (int\*) &pt2;** — покажчику **pt1** присвоюється адреса пам’ятi де розташована змінна **pt2**.

**Обмеження на застосування операції взяття адреси**:

* не можна визначати адресу літеральної константи (оскільки для неї не виділяється комірка пам’яті), тобто такий запис, як **vp = &345;** — неприпустимий;
* не можна визначати адресу результату арифметичного виразу, тобто запис **vp = &(x + y);** теж неприпустимий.

**Дозволені операції для змінних-покажчиків:**

* операція розіменування «\*»;
* операція взяття адреси «&»;
* операція присвоювання «=»;
* операції інкремент «++» і декремент « –-»;
* операції додавання «+» і віднімання «-»;
* операції відношення (порівняння) покажчиків однакового типу: «==», «!=», «<», «<=», «>», «>=».

У мові C++ масиви і покажчики зв’язані між собою: ***ім’я масиву визначається як покажчик-константа на початковий (нульовий)елемент масиву.*** Так, наприклад, при оголошенні одновимірного масиву у вигляді **int mas [20];** його ім’я **mas** – покажчик на адресу початкового елемента масиву **&mas[0]**.

Існує два способи доступу до елементів масиву:

* з використанням індексу елемента масиву, наприклад, **mas[2]** або **mas[i];**
* з використанням адресного виразу, тобто виразу з покажчиком на масив, наприклад, **\*(mas + 2)** або **\*(mas + і)**.

Ім’я покажчика на масив можна записати так:

**int mas [20];**  
**int \*ptr1;**  
**ptr1 = mas;**      *//ptr1* *= &mas[0];,*

тут вирази **&mas[0] і mas** — еквівалентні.

Оскільки в комп’ютері для масивів завжди є суцільний блок комірок пам’яті, в яких розташовуються їх елементи, то адресу наступного елемента **mas[1]** можна вказати шляхом збільшення покажчика на **1**, а саме:

**р = &mas[0];**

**р++;** *//р=р  + 1;*

Таким чином, адреса **і**-го елемента визначається як **р + і**. При цьому з урахуванням типу масиву і відведеної кількості байтів для кожного його елемента автоматично виконується операція збiльшення адреси, тобто:

**адреса х[і] = адреса х[0] + i\*sizeof (тип);** .

***Для покажчиків, які посилаються на елементи масивів різних типів, результат арифметичних операцій і операцій відношення невизначений.***

До двох покажчиків **р1 і р2**, що вказують на елементи одного масиву, застосовують операції відношення: «==», «!=», «<», «<=», «>», «>=». При цьому значення покажчиків розглядаються як цілі числа, а результат порівняння дорівнює **0** ( »неправда») або **1** («істина»). Так, відношення вигляду **р1<р2** є «істина», якщо **р1** указує на більш ранній елемент, ніж **р2**. Будь-який покажчик можна порівняти з нулем.

В арифметиці з покажчиками можна використовувати адресу неіснуючого «наступного за масивом» елемента. До покажчиків можна додавати або віднімати від них цілу величину.

В обох випадках результатом операції буде покажчик на вихідний тип, значення якого на вказане число елементів більше або менше вихідного. Тобто, якщо до покажчика **р** можна додати деяку цілу величину **n**, а саме: **р + n**, то цей вираз визначає ділянку об’єкта, що займає **n**-не місце після об’єкта, на який вказує **р**, при цьому **n** автоматично збільшується на коефіцієнт, що дорівнює відповідній довжині об’єкта. Наприклад, якщо **int** займає 4 байти, то цей коефіцієнт дорівнює чотирьом.

Допускається також операція віднімання покажчиків, що вказують на елементи одного масиву. Так, якщо **р1 < р2**, то **р2 – р1 + 1** — це число елементів масиву від **р1** до **р2** включно.

**Запитання**

1. Що таке масив та які існують різновиди масивів?
2. Як здійснюється звернення до елементів масивів?
3. Що таке рядки та значення елементів символьного типу?
4. Що являє собою масив символьного типу?
5. Як здійснюється введення символьних даних?
6. Як можна в програмі визначити рядок символів?
7. Як виконується порівняння даних символьного типу?
8. Як визначити кількість символів у рядку?
9. Як описуються дані типу структура?
10. Які типи полів може містити структура?
11. Назвіть функції визначення довжини рядка.
12. Назвіть функції додавання одного рядка або його частини до іншого рядка.
13. Яка функція використовується для включення рядка в рядок.
14. Яка функція використовується для вилучення символів із рядка.
15. Яким чином можна замінити частини рядка або увесь рядок?
16. Як обміняти зміст двох рядків?
17. Яка функція використовується для виділення частини рядка?
18. Як здійснити пошук позиції входження підрядка в рядок?
19. Як порівняти рядки або їхні частини?
20. Яка функція виконує перетворення рядка типу string у рядок типу char?