**ЛАБОРАТОРНА РОБОТА № 6.**

**Вказівники в мові програмування С++**

**6.1. Мета роботи**

Навчитися використовувати вказівники при написанні програм на мові С++.

# 6.2. Теоретичні відомості

6.2.1. Поняття вказівника

*Вказівник* (*pointer*) – особливий вид змінної, яка зберігає адресу об’єкту в пам'яті та таким чином вказує на нього.

Оголошення вказівника:

**<тип> \* <ідентифікатор>;**

“**\***” – астерікс (зірочка), декларує, що змінна є вказівником. Наприклад:

**іnt a = 5; // a has type (int) int \*p // p has type (int\*)**

Вказівники надають символічний спосіб роботи з адресами. Оскільки апаратні інструкції обчислювальних машин у великій мірі покладаються на адреси, вказівники дають змогу виражати дії в манері, близькій до машинного представлення. Така відповідність робить програми з вказівниками ефективними. Зокрема, вказівники пропонують ефективний метод маніпулювання масивами. Нагадаємо, що ім’я масиву це константний вказівник на його перший елемент. Вказівники відкривають широкі можливості для передачі аргументів у функції без необхідності їх копіювання.

6.2.2. Операції з вказівниками Операція взяття адреси:

Операція, що нерозривно пов'язана з вказівниками – унарна операція *взяття адреси*: “**&**”. Наприклад:

**p = &a;**

Тут в **p** записується адреса змінної **a**, тобто **p** починає вказувати на **a**. Результатом застосування операції “**&**” є адреса об’єкту в пам'яті. Результат має тип “вказівник” на тип змінної. Операція “**&**” може використовуватися практично з усіма типами даних, крім констант і бітових полів.

Операція розіменування вказівника:

Щоб отримати об’єкт в пам’яті, на який вказує вказівник необхідно його розіменувати. Для цього існує *операція розіменування вказівника* “**\***”.

Наприклад:

**int b = \*p; // now b equals a**

Не слід плутати оголошення вказівника, де астерікс декларує, що ідентифікатор має тип “вказівник” і операцію розіменування, яка використовується у виразах. Результат операції – вміст комірки пам'яті, на яку вказує **p**. Слід зазначити, що зі змінною (виразом) **\*p** можна працювати як зі звичайною змінною.

Наприклад:

**\*р += 3; // now a equals 8**

Операція присвоювання:

Операція *присвоювання* для вказівників аналогічна відповідній операції для інших типів даних. Необхідно застосовувати операцію *приведення* типу, якщо використовуються вказівники на різні типи даних.

Операція збільшення (зменшення) вказівника:

**E + і; E - і;**

Тут **Е** – змінна типу “вказівник”, a **і** – значення цілочисельного типу. Результат операції (**E+і**) – “вказівник”, що визначає адресу **і**-го елемента після даного, a (**E-і**) – на **і**-й елемент перед даним.

Операція складного присвоювання:

**E += і; E -= і;**

Тут **Е** – змінна типу “вказівник”, a **і** – значення цілочисельного типу. Ці операції аналогічні виразам (відповідно):

**E = E + і; E = E - і;**

Операції інкременту (декременту):

**Е++; Е--; ++E; --Е;**

Виконання даних операцій аналогічно відповідним операціям над цілочисельними типами, тобто вказівник буде зміщатися (збільшуватися або зменшуватися залежно від операції) на один елемент, фактично вказівник (адреса) зміниться на кількість байтів, що займає цей елемент у пам'яті.

Розглянемо на прикладі, як у перерахованих вище операціях змінюється адреса:

**#include <cstdio>**

**int main()**

**{**

**int a, \*pi = &a;**

**float f, \*pf = &f;**

**printf("pi = %p pf = %p\n", pi, pf);**

**pi++;**

**pf++;**

**printf("pi = %p pf = %p\n", pi, pf);**

**return 0;**

**}**

Можливий результат:

**pi = 00000047E5B3F524 pf = 00000047E5B3F564**

**pi = 00000047E5B3F528 pf = 00000047E5B3F568**

Насправді в результаті **pi++** вказівник зміниться на **sizeof(іnt)** байт, а при операції pf++ – на sizeof(float) байт. Взагалі, якщо оголошено:

**<тип> \*p;**

то операція **p = p + і**, де **і** – ціле, змінить **p** на **sіzeof(<тип>)\*(і** байт**)**. Операція індексування:

**E[і]**

Тут **Е** – змінна типу “вказівник”, a **і** – значення цілочисельного типу. Ця операція повністю аналогічна виразу **\*(E+і)**, тобто з пам'яті вибирається й використовується у виразі значення **і**-гo елемента масиву, адреса якого присвоєна вказівнику **Е**.

Згідно стандарту, постфіксна операція типу **E1[E2]** є еквівалентною до **(\*((E1) + (E2)))** – компілятор завжди сам заміняє всі операції індексування на операції з вказівниками при генеруванні об’єктного коду.

Операція віднімання вказівників:

**Е1 - Е2;**

Тут **El, E2** – змінні типу “вказівник”, причому вказівники на той самий набір даних і, природно, одного типу. Інакше операція безглузда. Тип результату залежить від компілятора і дорівнює кількості елементів, які можна розташувати в комірках пам'яті з **Е2** по **Е1**. Тип результату визначений в стандартній бібліотеці **<stddef.h>** як макрос **ptrdiff\_t** і зазвичай це **int** або **long**.

Операції відношення:

**Е1 == Е2; Е1 >= Е2; Е1 > Е2;**

**Е1 < Е2; Е1 <= Е2; E1 != E2;**

Результат всіх операцій має тип **іnt**. Результат операції “**==**” буде дорівнювати одиниці (true), якщо **Е1** і **Е2** указують на той самий елемент в оперативній пам'яті. У протилежному випадку результат буде дорівнювати нулю (false). Результат операції “**>=**” одиниця (true), якщо об'єкт **\*Е1** розташований у пам'яті в старших адресах, ніж об'єкт **\*Е2**. Результатом операції “**<**" буде одиниця (true), якщо об'єкт **\*Е1** розташований в молодших адресах, ніж об'єкт **\*Е2**. Крім того, будь-який вказівник (адреса) може бути перевірений на рівність **(==)** або нерівність (**!=**) зі спеціальним значенням **NULL** (**NULL == 0**) nullptr (з версії С11). Функції виділення пам'яті повертають **NULL** з появою яких-небудь помилок. Тому порівняння з **NULL** часто використовується для визначення помилок виділення пам'яті:

6.2.3. Особливий тип вказівника – **voіd\***

Вказівник типу **voіd\*** визначає місце в оперативній пам'яті (адреса деякого байту), але не містить інформації про тип об'єкту. До використання значення, що перебуває по цій адресі, обов'язково повинна бути виконана операція приведення вказівника до деякого типу, тому що в протилежному випадку компілятору буде невідома довжина поля пам'яті, яке використовується в операції. До вказівника типу **voіd\*** застосовуються наступні операції:

**=** – *просте присвоювання*;

**==, !=, >, <, <=, >=** – *операції порівняння*.

Приклад використання вказівника без типу (**voіd\***) для демонстрації розміщення в пам'яті довгих даних (**long іnt**). При відображенні на екрані вміст байтів пам'яті виводиться в нормальній послідовності:

|  |
| --- |
| **#include <stdio.h>**  **int main()**  **{ long x = 0x12345678; // 78563412 in memory void \*p = &x; // first byte**  **printf("char = %x\n",\*(char\*) p);// first two bytes**  **printf("short = %x\n",\*(short\*)p);**  **// all four bytes**  **printf("long = %x\n",\*(long\*) p);**  **return 0;**  **}** |

Результат виконання, при умові що **sizeof(char) = 1**, **sizeof(short)= 2**, **sizeof(long) = 4** (залежить від компілятора):

**char = 78**

**short = 5678**

**long = 12345678**

6.2.4. Вказівник на **char**

Оскільки мова C++ підтримує С-type, для роботи з ними зазвичай використовуються вказівники типу **char\***. Якщо стрічкова константа використовується для ініціалізації вказівника типу **char\***, то адреса першого символу змінної буде початковим значенням вказівника. Наприклад:

**char \*str = "cat";**

Тут описується тільки вказівник **str**, і вказівник одержує початкове значення, рівне адресі першого елемента (символу **'с'**) стрічкової константи. Компілятор виділить пам'ять як для рядка (чотири байти), так і для розміщення значення вказівника.

Якщо стрічкова константа використовується в тих місцях виразу, де дозволяється застосовувати вказівник, компілятор підставляє у вираз замість константи адресу її першого символу. Наприклад:

**char \*str;**

**str = "cat";**

Рядок можна ввести за допомогою функції **scanf**, використовуючи вказівник (для успішної компіляції вихідного коду з функцією **scanf** у Microsoft Visual Studio додатково додайте макрокоманду **#define \_CRT\_SECURE\_NO\_WARNINGS**):

**char \*str, rt[20], s[15];**

**str = s;**

**scanf("%s", str);**

**scanf("%s", rt);**

Ввести рядки можна й так:

**scanf("%s", &str[0]);**

**scanf("%s", &rt[0]);**

Також можна створити масив вказівників типу **char**:

**char \*strl[10];**

Ініціалізацію масиву рядків і масиву вказівників можна виконати в такий спосіб:

**char fg[][6] = {"one", "two", "three"};**

**char \*strn[2];**

**strn[0] = "OMEGA";**

**strn[1] = "1234567";**

**char \*strm[2] = {"DELTA","9876543"};**

Ввести рядки можна наступними операторами:

**char sd[4][30];**

**for(int i = 0; i < 4; i++)**

**scanf("%s", sd[i]);**

**char \*jk[4], s[4][30];**

**for(int i = 0; i < 4; i++){**

**jk[i] = s[i];**

**fgets(jk[i], 25, stdin);**

**}**

Перевага використання масивів вказівників у тому, що можна оперувати не самими об'єктами, а тільки їхніми адресами, що дає значний виграш у швидкості виконання програми.

Як приклад роботи з рядками розглянемо програму, де сортується масив рядків, використовуючи вказівники на них. Ознакою кінця введення рядків буде введення порожнього рядка:

|  |
| --- |
| **#include <stdio.h>**  **int main()**  **{**  **char str[25][25], \*ptrs[25];**  **puts("Please input few words, "**  **"input empty line to stop");**  **int length;**  **for(length = 0; ; ++length)**  **{**  **gets(str[length]);**  **ptrs[length] = str[length];**  **if(!str[length][0])break;**  **}**  **for(int j = 0; j < length; ++j)**  **for(int g = 0; g < length - 1; ++g)**  **{**  **char \*c1 = ptrs[g];**  **char \*c2 = ptrs[g + 1];**  **int flag;**  **for(; \*c1 == \*c2; ++c1, ++c2)**  **if(!\*c1)**  **{**  **flag = 0;**  **break;**  **}**  **flag = \*c1 - \*c2;**  **if(flag > 0)**  **{**  **char \*p = ptrs[g];**  **ptrs[g] = ptrs[g + 1];**  **ptrs[g + 1] = p;**  **}**  **}**  **puts("Sorted words:");**  **for(int j = 0; j < length; ++j) puts(ptrs[j]);**  **return 0;**  **}** |

Результати:

**Please input few words, input empty line to stop**

**Kiev**

**Warsaw**

**London**

**Paris**

**Rome**

**Sorted words:**

**Kiev**

**London**

**Paris**

**Rome**

**Warsaw**

При вирішенні завдання використовувався як двомірний масив, так і масив вказівників на рядки. Для того щоб поміняти рядки місцями, досить поміняти місцями вказівники на дані рядки в масиві вказівників. Масив вказівників сортується так, щоб перший елемент вказував на “наймолодший” за алфавітом рядок, а останній – на самий “найстарший” рядок. При введенні рядків кожний рядок порівнюється з нульовим для перевірки умови закінчення вводу.

Розглянемо приклад – обчислення довжини рядка:

|  |
| --- |
| **#include <stdio.h> #include <string.h> int main()**  **{ char str[] = "1, 2, 3", \*s;**  **int l1 = 0; while(str[++l1]);**  **int l2 = 0; for(s = str; \*s++;); l2 = s-str-1;**  **int l3 = strlen(str);**  **printf("%d %d %d\n", l1, l2, l3);**  **return 0;**  **}** |

Результати:

**7 7 7**

6.2.5. Зв'язок між вказівниками й масивами

У мові C при обробці елементів масиву зручно використовувати вказівник на цей масив. Будь-який доступ до елемента масиву, що здійснюється операцією індексування, може бути виконаний і за допомогою вказівника. Фактично, компілятор заміняє всі операції індексування на операції з вказівниками, при генеруванні об’єктного коду. Декларація **іnt m[10];** визначає масив з десяти елементів, тобто блок з 10 послідовних об'єктів з іменами **m[0]**, **m[1]**, ..., **m[9]**. Запис **m[і]** є зміщенням адреси до і-го елемента масиву. Одночасно з виділенням пам'яті для десяти елементів типу **іnt** визначається значення константного вказівника **m**. Значення константного вказівника **m** дорівнює адресі елемента **m[0]**. Значення константного вказівника **m** змінити не можна, але його можна присвоїти іншому вказівнику й змінювати вже його значення, а також використовувати у виразах:

**\*(m + j + 2) = 5; \*(m + 3) = 1;**

Якщо **ра** є вказівник на **іnt**, тобто визначений як:

**іnt \*ра;**

то в результаті присвоювання:

**ра = &m[0];**

**ра** буде вказувати на нульовий елемент **m**; інакше кажучи, **ра** буде містити адресу елемента **m[0]**. Тепер присвоювання:

**іnt х = \*ра;**

буде копіювати вміст **m[0]** у **х**. Якщо **ра** вказує на деякий елемент масиву, то **ра + 1** за визначенням вказує на наступний елемент, **ра + і** – на **і**-й елемент після **pa**, a **pa - і** – на **і**-й елемент перед **ра**.

Оскільки ім'я масиву є не що інше, як адреса його початкового елемента, то присвоювання **ра = &m[0];** можна реалізувати й так:

**pa = m;**

З іншого боку, якщо **pa** – вказівник, то у виразах його можна використовувати й з індексом, тобто **pa[і]**. Але між ім'ям масиву й вказівником, що має значення адреси масиву, є істотне розходження. Вказівник це змінна, тому можна записати вираз:

**іnt m[5];**

**іnt \*pa = m; pa++;**

А ім'я масиву – це вказівник-константа, тобто записи:

**іnt x[5], \*pa x, m[5];**

**m = pa; m++; // Error**

Для двовимірних масивів ім'я є вказівником-константою на масив вказівників-констант. Елементами масиву вказівників є вказівники-константи на початок кожного з рядків масиву: тому, при використанні вказівників "точкою відліку" може бути як найперший елемент масиву, так і перший елемент кожного з рядків, тобто можуть використовуватися як вказівникконстанта, що задається ім'ям масиву, так і вказівники на рядки масиву.

Наприклад, для массиву:

**char str[3][50] = {"One", "Two", "Three"};**

вираз:

**putc(str[1][3]);**

може бути записане у вигляді:

**putc(\*(str[1] + 3));**

або:

**putc(\*(\*(str + 1) + 3));**

Якщо існують два вказівники **p1** і **р2**, що посилаються на той самий масив **mas**, то до них можна застосувати наступні операції:

**pl++;**

Установлює вказівник **p1** на наступний елемент масиву mas.

**р2--;**

Установлює вказівник **р2** на попередній елемент масиву mas.

|  |  |
| --- | --- |
| **pl+=і;** |  |
| Установлює вказівник **p1** | на і-й елемент після **p1**. |
| **p2-=і;** |  |
| Установлює вказівник **р2** | на і-й елемент перед **р2**. |
| **р2-р1;** |  |

Дає число елементів між **p1** і **р2**.

**р1 == р2;**

**р1 != р2;**

**p1 < p2;**

**pl > p2;**

**р1 <= р2;**

**р1 >= р2;**

Виконують порівняння вказівників.

Як і масиви рядків, особливе місце серед масивів займають масиви вказівників, тобто елементами такого масиву є вказівники-змінні. Найбільш часто вони використовуються для компактного розташування в пам'яті рядків тексту, структурних змінних та інших великих об'єктів даних. Формат опису масиву вказівників має вигляд:

**<тип> \*<ім'я> [<розмір1>][<розмір2>]...;**

Наприклад :

**char \*name[10];**

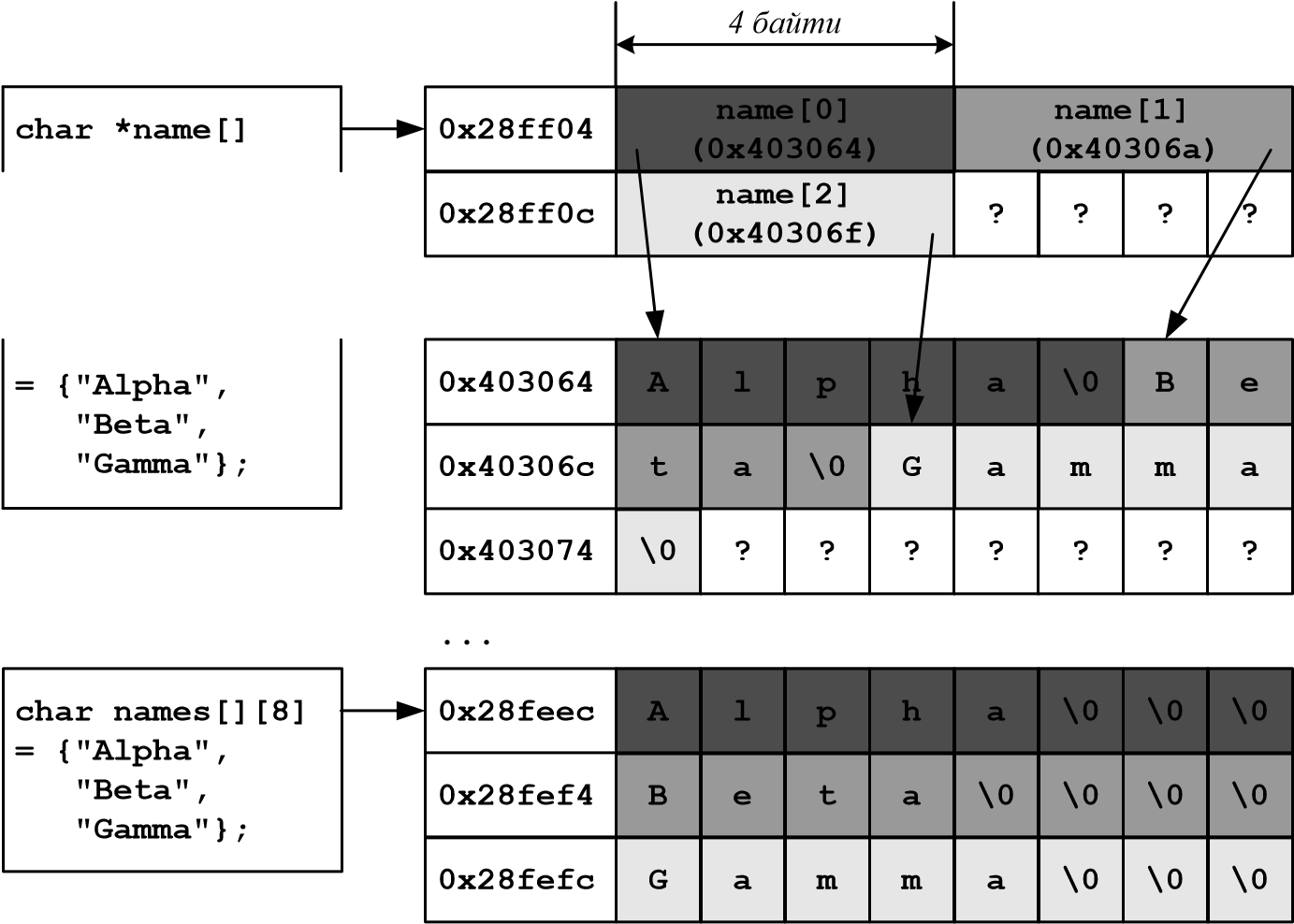
Кожний елемент масиву **name** має тип **char\*** (є вказівником на рядок). При визначенні масиву вказівників він може бути проініціалізований:

**char \*name[] = {** **"Alpha", "Beta", "Gamma"};**

Компілятор резервує місце для трьох вказівників, кожний з яких одержує початкове значення, рівне адресі початку в пам'яті відповідного рядка. Існує велике розходження в розташуванні в пам'яті масиву вказівників і подібного йому двовимірного масиву типу **char**. Наприклад :

**char names[][8] = {** **"Alpha", "Beta", "Gamma"};**

У пам'яті масиви **name** і **names** можуть виглядати так (рис. 6.1).



*Рис. 6.1 Розміщення рядків символів у пам’яті при ініціалізації*

Перевага використання вказівників у тому, що застосовуючи їх, можна маніпулювати не самими даними, а їхніми адресами, що дає виграш у швидкості виконання програми.

6.2.6. Використання вказівників при роботі з багатомірними масивами При оголошенні масиву:

**float fmas[2][3];**

**fmas** – це вказівник-константа на матрицю. Вираз **frnas[0]** і **fmas[1]**, у свою чергу, також є вказівниками-константами на нульовий і перший рядки. Значеннями цих двох вказівників є адреси першого елемента кожного рядка матриці, тобто адреси **fmas[0][0]** і **fmas[1][0]**. Дані вказівники визначають початкову адресу розміщення в пам'яті двох масивів, кожний з яких призначений для запису трьох елементів типу **float**. Слід зазначити, що фактично в пам'яті ніякого масиву вказівників з елементами **fmas[0]** і **fmas[l]** компілятор не створює. Взаємозв'язок цих елементів можна зобразити в такий спосіб (рис. 6.2).

**float fmas[2][3] = {{0.0, 0.1, 0.2},**

**{1.0, 1.1, 1.2}};**

**fmas**

**[**

**0**

**][**

**0**

**]=**

**0**

**.**

**0**

*4*

*байти*

**0**

**x**

**28**

**ff**

**08**

**fmas**

**fmas**

**[**

**0**

**][**

**1**

**]=**

**0**

**.**

**1**

**fmas**

**[**

**0**

**][**

**2**

**]=**

**0**

**.**

**2**

**fmas**

**[**

**1**

**][**

**0**

**]=**

**1**

**.**

**0**

**fmas**

**[**

**1**

**][**

**1**

**]=**

**1**

**.**

**1**

**fmas**

**[**

**1**

**][**

**2**

**]=**

**1**

**.**

**2**

**0**

**x**

**28**

**ff**

**10**

**0**

**x**

**28**

**ff**

**18**

**fmas**

**[**

**0**

**]**

**(**

**0**

**x**

**28**

**ff**

**08**

**)**

**fmas**

**[**

**1**

**]**

**(**

**0**

**x**

**28**

**ff**

**1**

**c**

**)**

*Рис. 6.2 Розміщення в пам’яті двовимірного масиву елементів типу* **float**

Для установки вказівника на деякий рядок матриці використовуються вирази виду **fmas[і]** або **\*(fmas+і)**. Вибірка самих значень елементів матриці реалізується виразами **fmas[і][j],** або **\*(\*(fmas+і)+j)**, або **(\*(fmas+і))[j]**. Оскільки, елементи багатомірних масивів у пам'яті розміщаються послідовно, то можна звертатися до будь-якого елемента такого масиву, використовуючи вказівник і один індекс.

6.2.7. Вказівник на вказівник

У мові C можна оголошувати змінні, що мають тип “*вказівник на вказівник*”. Ознакою такого оголошення є повторення астеріксу при оголошенні змінної. Число астеріксів визначає число “рівнів” вказівника. Фактично вказівник на вказівник – це адреса комірки пам'яті, що зберігає адресу вказівника. При визначенні вказівник на вказівник може ініціалізуватися.

Наприклад:

**іnt data = 5; іnt \*ptr = &data; іnt \*\*pptr = &ptr; іnt \*\*\*ppptr = &pptr;**

Для отримання значення **data** всі нижченаведені записи рівносильні:

**ptr[0] \*ptr pptr[0][0]**

**\*\*pptr ppptr[0][0][0] \*\*\*ppptr**

Нехай оголошено масив:

**іnt хi[] = { 1,2,3,4,5,6,7}; іnt \*pxі1[] = { xі, xі+1, xі+2, xі+3, xі+4, xі+5, xі+6, xі+6, NULL}; іnt \*\*pxі2 = pxі1;**

Взаємозв'язок між цими змінними можна представити як (рис. 6.3).

**xi**

**==**

**&**

**xi**

**[**

**0**

**]**

**1**

**pxi**

**1**

**==**

**&**

**pxi**

**1**

**[**

**0**

**]**

**pxi**

**2**

**==**

**&**

**pxi**

**2**

**[**

**0**

**]**

**2**

**3**

**4**

**5**

**6**

**7**

**&**

**xi**

**[**

**0**

**]**

**&**

**xi**

**[**

**1**

**]**

**&**

**xi**

**[**

**2**

**]**

**&**

**xi**

**[**

**3**

**]**

**&**

**xi**

**[**

**4**

**]**

**&**

**xi**

**[**

**5**

**]**

**&**

**xi**

**[**

**6**

**]**

**&**

**xi**

**[**

**6**

**]**

**NULL**

*Рис. 6.3 Розміщення рядків символів у пам’яті при ініціалізації*Приклад звертання до елементів масиву з використанням вказівників:

|  |
| --- |
| **#include <stdio.h> int main()**  **{**  **int x[]={1, 2, 3, 4, 5, 6},**  **k;**  **int \*px1[]={**  **x, x+1, x+2, x+3, x+4, \**  **x+5,x+6, x+6, NULL},**  **\*ptr;**  **int \*\*px2 = px1;**  **px2+=3; k=\*\*px2; printf("%d ", k);**  **ptr=\*++px2; k= \*ptr; printf("%d ", k);**  **px2=px1; ptr=\*++px2; k=\*ptr; printf("%d ", k);**  **++\*px2; k=\*\*px2; printf("%d ", k);**  **px2=px1; k=\*++\*px2; printf("%d ", k);**  **return 0;**  **}** |

Результати виконання:

**4 5 2 3 2**

Приклад роздруку рядків масиву:

|  |
| --- |
| **#include <stdio.h>**  **int main()**  **{**  **char \*str[] = {**  **"first", "second", "third", "fourth", 0};**  **char \*\*pptr = str; while(\*pptr)**  **puts(\*pptr++); return 0;**  **}** |

Результати виконання:

**first second third fourth**

6.2.8. Приклади програм з використанням вказівників

З рядка виділити слова, записати їх у масив і вивести масив слів:

|  |
| --- |
| **#include <stdio.h> #include <conio.h>**  **int main()**  **{ char x[100], y[20][100], \*z;**  **for(;;)**  **{ int words = 0;**  **puts("Press 'q' to exit, "**  **"press 'a' to continue");**  **switch(getch())**  **{**  **case 'q': return 0;**  **case 'a':**  **puts("Please input string");**  **gets(x);**  **z = x;**  **for(words = 0; \*z; ++words)**  **{**  **int i;**  **for(i=0; \*z != ' ' &&**  **\*z; ++z, ++i)**  **y[words][i] = \*z;** |
| **y[words][i] = '\0';**  **if(\*z == ' ') ++z;**  **}**  **for(int i=0; i<words; ++i)**  **puts(y[i]);**  **break;**  **}**  **}**  **return 0;**  **}** |

Результати виконання:

|  |
| --- |
| **Press 'q' to exit, press 'a' to continue**  **Please input string**  **Time is money Time**  **is**  **money**  **Press 'q' to exit, press 'a' to continue** |

Використання складних оголошень:

|  |
| --- |
| **#include <stdio.h>**  **int main()**  **{**  **int i;**  **char \*c[]={"1a","2b","3c","4d","5e"};**  **char \*\*cc[5];**  **char \*\*\*ccc[5];**  **char \*\*\*\*cccc[5];**  **char \*\*\*\*\*ccccc[5];**  **char \*\*\*\*\*\*cccccc[5];**  **for(i=0;i<5;i++)**  **{**  **printf("c[%d]=%s ",i,c[i]);**  **cc[i]=&c[i];**  **ccc[i]=&cc[i];**  **cccc[i]=&ccc[i];**  **ccccc[i]=&cccc[i];**  **cccccc[i]=&ccccc[i];**  **printf("cc=%s "**  **"ccc=%s "**  **"cccc=%s "**  **"ccccc=%s "** |

**"cccccc=%s\n",**

**\*cc[i],**

**\*\*ccc[i],**

**\*\*\*cccc[i],**

**\*\*\*\*ccccc[i],**

**\*\*\*\*\*cccccc[i]);**

**} return 0;**

**}**

Результати виконання:

**c[0]=1a cc=1a ccc=1a cccc=1a ccccc=1a cccccc=1a c[1]=2b cc=2b ccc=2b cccc=2b ccccc=2b cccccc=2b c[2]=3c cc=3c ccc=3c cccc=3c ccccc=3c cccccc=3c c[3]=4d cc=4d ccc=4d cccc=4d ccccc=4d cccccc=4d c[4]=5e cc=5e ccc=5e cccc=5e ccccc=5e cccccc=5e**

6.2.9. Динамічне виділення пам`яті

При вирішенні прикладних задач дуже часто виникає потреба працювати з об’єктами, що виникають та зникають у процесі виконання програми динамічно. Наприклад наперед не відомо скільки рядків тексту введе користувач в програму для їх деякого подальшого опрацювання. Можна виділити наперед деякий великий буфер пам’яті і працювати з ним. Альтернативним варіантом є використання засобів динамічного виділення пам’яті. При цьому, пам'ять виділяється з вільної області в міру потреби й повертається назад, тобто звільняється, коли необхідність у ній зникла. Область вільної пам'яті, доступної для виділення, перебуває між областю пам'яті, де розміщається код програми, і стеком, де розміщують всі локальні та глобальні змінні, значення констант, масиви тощо. Ця область називається *купою* (*heap*).

Для виділення динамічного масиву і роботи з ним використовуються окремі форми операторів new і delete: new[] і delete[].

**#include <iostream>**

**int main()**

**{**

**std::cout << "Enter a positive integer: ";**

**int length;**

**std::cin >> length;**

**int \*array = new int[length]; // використовуємо оператор new[] //для виділення масиву. Змінна length не обов'язково повинна //бути константою!**

**std::cout << "I just allocated an array of integers of length " << length << '\n';**

**array[0] = 7; // присвоюємо елементу під індексом 0 значення 7**

**delete[] array; // використовуємо оператор delete[]**

**// для звільнення виділеної масиву пам'яті**

**array = 0; // використовуйте nullptr замість 0 в C++11**

**return 0;**

**}**

Приклад програми, що виводить стрічку випадкових символів (за допомогою функції бібліотеки **<stdlib.h>**: **rand()**), довжиною, яку вказує користувач. При цьому, пам’ять під стрічку виділяється динамічно:

|  |
| --- |
| **#define \_CRT\_SECURE\_NO\_WARNINGS**  **#include<cstdio>**  **#include<cstdlib>**  **int main()**  **{**  **int n;**  **char \*buffer;**  **printf("What length of the string "**  **"do you want?\n");**  **scanf("%d", &n);**  **buffer = new char [n];**  **if(buffer == NULL)**  **return -1;**  **for (int i = 0; i < n; ++i)**  **buffer[i] = rand() % 26 + 'a';**  **buffer[n] = '\0';**  **printf ("Random string: %s\n", buffer);**  **delete[] buffer;**  **return 0;**  **}** |

Результати:

**What length of the string do you want?**

**15**

**Random string: nwlrbbmqbhcdarz**

# 6.3. Контрольні запитання

1. Що таке вказівник?
2. Які оператори для роботи з вказівниками ви знаєте?
3. Який зв’язок між вказівником та масивом?

# 6.4. Лабораторне завдання

1. Навчитися використовувати вказівники при написанні програм на мові С++.
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.

**6.5. 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.

# 6.6. Індивідуальні завдання

Завдання 1

З клавіатури вводиться динамічний рядок, виконати наведене нижче завдання. При доступі до елементів використовувати вказівники.

1. Рядок містить символи нуля та одиниці, підрахувати кількість нулів та одиниць.
2. Порахувати скільки разів входять літери **'Р'** та **'S'**.
3. Рядок містить цілі числа, знаки арифметичної дії та дужки, перевірити, чи виконується баланс дужок **'('** та **')'**.
4. Рядок містить символи нуля та одиниці, перевірити, чи він симетричним відносно середини.
5. Перевірити, чи входить у нього цифри 5 та 7.
6. Порахувати кількість слів.
7. Перевірити, чи зустрічається у рядку подвоєння літер.
8. Порахувати кількість знаків пунктуації.
9. Порахувати скільки разів входить буквосполучення **"ED"** або **"ed"**.
10. Порахувати кількість слів, що починаються з літери **'a'**.
11. Визначити кількість входжень знаку **'\'** після появи знака **'#'**.
12. Перевірити, чи сьома з кінця літера рядка є літерою **'S'** або **'W'**.
13. Перевірити, чи входить літера **'S'** та через один символ є цифра **'1'**.
14. Перевірити, чи є слова, що починаються з **'a'** та закінчуються **'s'**.
15. Сформувати динамічний рядок, вивести його на друк, обчислити скільки разів у першу половину рядка входять літера **'S'**.
16. Перевірити, чи починаються слова нових речень з великої літери.

Завдання 2

Виконати наведене нижче завдання:

1. Задано текст. Створити масив вказівників на окремі абзаци. Посортувати їх за довжинами. Вивести посортовані абзаци на екран.
2. Задано текст. Створити масив вказівників на окремі речення. Посортувати їх за кодами літер. Вивести посортовані речення на екран.
3. Задано текст. Створити масив вказівників на окремі речення. Посортувати їх за довжинами. Вивести посортовані речення на екран.
4. Задано текст. Створити масив вказівників на окремі речення. Вивести речення у зворотньому порядку.
5. Задано текст. Створити масив вказівників на окремі речення. Посортувати кожне друге речення за кодами літер. Вивести посортовані речення на екран.
6. Задано речення. Створити масив вказівників на окремі слова. Посортувати їх за кодами літер. Вивести посортовані слова на екран.
7. Задано речення. Створити масив вказівників на окремі слова. Посортувати їх за довжинами. Вивести посортовані слова на екран.
8. Задано речення. Створити масив вказівників на окремі слова. Вивести слова у зворотньому порядку.
9. Задано речення. Створити масив вказівників на окремі слова. Посортувати кожне друге слово за кодами літер. Вивести посортовані слова на екран.
10. Задано слово. Створити масив вказівників на окремі літери. Посортувати їх за кодами літер. Вивести на екран.
11. Задано слово. Створити масив вказівників на окремі літери. Вивести у зворотньому порядку.
12. Задано слово. Створити масив вказівників на окремі літери. Посортувати кожну другу літеру за кодами літер. Вивести посортовані літери на екран.
13. Задано текст. Створити масив вказівників на окремі знаки пунктуації. Для кожного, вивести частину тексту від нього і до наступного.
14. Задано текст. Створити масив вказівників на окремі слова, що є числами. Посортувати їх за кодами за кодами літер. Вивести на екран посортовані числа.
15. Задано текст. Створити масив вказівників на окремі слова, що містять великі літери. Посортувати їх за кодами літер. Вивести на екран посортовані слова.
16. Задано текст. Створити масив вказівників на окремі слова, що містять тільки малі літери. Посортувати їх за кодами літер. Вивести на екран посортовані слова.