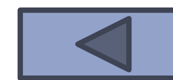
Тема 6: Покажчики

Вихід

ПЛАН НА СЕМЕСТР:

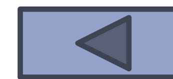
Семестр	другий
Лекції, годин	18
Лабораторні заняття, годин	22
Самостійна робота, годин	80
Всього годин	120
Всього кредитів ECTS	4
Форма підсумкового контролю	екзамен



Вихід

ОЦІНЮВАННЯ

Бали	Результат навчання, що оцінюється
6	Покажчики і масиви
6	Перелічуваний тип даних, структури, об’єднання
6	Перевантаження операторів
6	Перевантаження функцій
6	Робота з класами. Реалізація принципу інкапсуляції. Конструктори та деструктори
6	Управління доступом до компонентів класу. Одиночна спадкоємність. Віртуальні функції. Віртуальні деструктори.
6	Дружні класи та функції (friend). Правила доступу до членів класу. Поліморфізм
6	С++-система вводу-виводу
10	Контрольна робота
12	Індивідуально-розрахункова робота
30	Екзамен
100	Разом



ДЖЕРЕЛА:

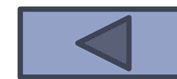
1. Бублик В.В. Об’єктно-орієнтоване програмування: [Підручник] / В.В. Бублик. – К.: ІТ-книга, 2015. – §2.4 С. 51 – 71.
2. Вступ до програмування мовою С++. Організація обчислень : навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставровський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – §1.2 С. 8 -12.
3. Зубенко В.В., Омельчук Л.Л. Програмування. Поглиблений курс. – К.:Видавничо-поліграфічний центр “Київський університет”, 2011. – §3.5 С. 337 – 349.
4. Schildt Herbert C++ from the Ground Up. - Third Edition. - McGraw-Hill/Osborne, 2003. – Chapter 6. P. 105 – 126.
5. Stroustrup Bjarne The C++ programming language / Bjarne Stroustrup. — Fourth edition. - Pearson Education, Inc, 2013. – §7.2 – 7.7 P. 172 – 198.

Вихід

ЗМІСТ

1. Поняття покажчика та посилання
2. Оператори, що використовуються з покажчиками
3. Операції з покажчиками.
4. Покажчики й масиви
5. Покажчики й рядкові літерали

Покажчики ≡ вказівники ≡ указівники



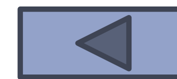
ВСТУП

При запуску програми операційна система завантажує її в деяку частину пам'яті.

Ця пам'ять, яка використовується програмою, розділена на кілька частин, кожна з яких виконує певне завдання.

Одна частина містить код, інша використовується для виконання звичайних операцій (відстеження функцій, створення і знищення глобальних і локальних змінних тощо).

Велика частина доступної пам'яті комп'ютера просто знаходиться в очікуванні запитів на виділення від програм - її називають **купою**.



Вихід

Поняття «покажчик»

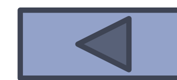
Означення: Покажчики – це змінні, можливими значеннями яких є адреси пам’яті.

Розрізняють покажчики на **об’єкти, функції та загального призначення типу void***. Останні ще називають **нетипізованими**.

Покажчики застосовуються у двох різних напрямах:

1) вони забезпечують прямий доступ до об'єктів програмних даних (як статичних, так і динамічних) і тим самим – можливість "ручного" керування пам'яттю;

2) можливості адресної арифметики й непрямої адресації використовують для оптимізації коду програми.



Вихід

Поняття «покажчик»

Формат оголошення змінної-покажчика:

тип *ім'я_змінної;

У 32-розрядних середовищах покажчик будь-якого типу займає чотири байти, у 64-розрядних – вісім байтів.

- **Базовий тип покажчика визначає тип даних, на які він буде посилатися**.
- Наприклад:

```
int *ip;
// покажчик на цілочисельне значення
double *dp;
// покажчик на значення типу double
```

Вважається коректною обов’язкова ініціалізація покажчика під час його опису.

Вихід

Існують такі способи ініціалізації покажчиків:

1. Привласнення покажчику адреси існуючого об’єкта:

- за допомогою операції отримання адреси:
- за допомогою значення іншого покажчика, що вже ініціалізований:
- за допомогою імені масиву або функції, які трактуються як адреса:

2. Привласнення покажчику адреси області пам’яті в явному вигляді.

3. Привласнення порожнього значення.

4. Виділення ділянки динамічної пам’яті і привласнення її адреси покажчику

- за допомогою операції new:
- за допомогою функцій malloc(), calloc(), realloc():

Вихід

Існують такі способи ініціалізації покажчиків:

1. Привласнення покажчику адреси існуючого об’єкта:

- **за допомогою операції отримання адреси:**

int a = 5; // ціла змінна

int *p = &a; // у покажчик записується адреса a

int *p(&a); // те ж саме іншим способом;

за допомогою значення іншого покажчика, що вже ініціалізований:

int* r = p;

- **за допомогою імені масиву або функції, які трактуються як адреса:**

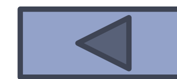
int b[10]; // масив

int *t = b; // привласнення адреси початку масиву

void f(int a) { /* ... */ } // визначення функції

void (*pf)(int); // покажчик на функцію

pf = f; // привласнення адреси функції.



Вихід

Існують такі способи ініціалізації покажчиків:

2. Привласнення покажчику адреси області пам’яті в явному вигляді:

```
char *vp = (char *)0xB8000000;
```

тут 0xB8000000 – шістнадцятирічна константа, (char *) – операція приведення типу: константа перетвориться до типу "покажчик на char".

3. Привласнення порожнього значення:

```
int *suxp = NULL;
int *rulezp = 0;
```

Оскільки гарантується, що об’єктів з нульовою адресою немає, порожній покажчик можна використовувати для перевірки, посилається покажчик на конкретний об’єкт чи ні.

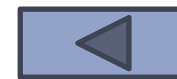
Існують такі способи ініціалізації покажчиків:

4. Виділення ділянки динамічної пам’яті і привласнення її адреси покажчику за допомогою операції new:

```
int *np = new int; //1
int *mp = new int (20); // 2
int *qp = new int [20]; // 3
```

за допомогою функції malloc():

```
int *up = (int *)malloc(sizeof(int)); // 4
```



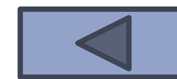
ФУНКЦІЇ ВИДІЛЕННЯ ПАМ'ЯТІ

- Прототипи функцій для роботи з пам'яттю містяться в бібліотеці <cstdlib>.
- Для виділення пам'яті служать функції
- **void *malloc(size_t n)** – виділяє в купі поле пам'яті, достатнє для розміщення об'єкта довжиною n байтів, і повертає його безтипову адресу, яка може бути явно перетворена до будь-якого потрібного типу. Якщо в купі недостатньо місця, то функція повертає значення NULL.
- **void *calloc(size_t n, size_t m)** – виділяє в купі поле пам'яті, достатнє для розміщення масиву довжиною n із компонентами довжиною m байтів, і повертає його безтипову адресу, яка може бути явно перетворена до типу компонент з урахуванням вирівнювання. При виділенні пам'яті її комірки обнулюються.
- **void *realloc(void* ptr, size_t n)** – змінює розмір поля, виділеного раніше однією зі стандартних функцій зі збере

ФУНКЦІЇ ВИДІЛЕННЯ ПАМ'ЯТІ

- Звільнення пам'яті, виділеної за допомогою оператора **new**, повинно виконуватися за допомогою **delete**.
- Функція **void free(char *ptr)** звільняє область купи, розподілену раніше функціями **malloc()** або **realloc()**. Аргументом її є покажчик, що повертається однією із цих функцій.
- Виклик **free(NULL)** не здійснює жодних дій.
- Функція **void cfree(char *ptr)** робить те саме, але для покажчиків, що повертається функцією **calloc()**.
- В усіх випадках змінна-покажчик зберігається і може ініціалізуватися повторно.
- Наведені вище динамічні змінні знищуються таким чином:

delete np; delete mp; delete [] qp; free (up);



Поняття «покажчик»

- За допомогою комбінацій зірочок, круглих і квадратних дужок можна описувати складені типи і покажчики на складені типи, наприклад:

  int *(*p[20])();

оголошується масив з 20 покажчиків на функції без параметрів, що повертають покажчики на int.

- За замовчуванням квадратні і круглі дужки мають однаковий пріоритет, більший, ніж зірочка, і розглядаються зліва направо.

Вихід

Поняття «покажчик»

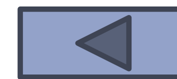
При інтерпретації складних описів необхідно дотримуватися правила із середини назовні:

1) якщо праворуч від імені є квадратні дужки, то це масив, якщо дужки круглі – функція;

2) якщо зліва є зірочка, це покажчик на проінтерпретовану раніше конструкцію;

3) якщо справа зустрічається закриваюча кругла дужка, необхідно застосувати наведено вище правило усередині дужок, а потім переходити назовні;

4) в останню чергу інтерпретується специфікатор типу.

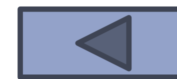


Посилання (reference)

- це видозмінена форма покажчика, яка використовується в якості псевдоніму (другого імені) змінної. У зв’язку з цим посилання не потребують додаткової пам’яті. Для визначення посилання використовують символ & (амперсанд), який ставиться перед змінною-посиланням.

Між посиланням та покажчиком можна виділити такі основні відмінності:

1) при оголошенні посилання має бути обов’язково ініціалізоване, а покажчик не обов’язково;
2) після оголошення посилання та його ініціалізації, цьому посиланню не можна присвоювати адреси інших об’єктів. Покажчик може змінювати свої значення скільки завгодно;
3) покажчику можна присвоювати нульове значення при оголошенні. Посилання не може мати нульового значення.
4) при оголошенні масиву посилань компілятор повідомить про помилку. Оголошувати масив покажчиків дозволяється.



Вихід

```
/*Програма 1. Демонстрація посилання */
#include <iostream>
using namespace std;
int main() {
   int t = 13;
   int& ref = t;// ініціалізація посилання на t,  ref є синонімом імені t
   setlocale(LC_ALL, "ukr");
   cout  <<  "Початкове  значення  t:  "  <<  t << endl;
   // виводить 13
   ref += 10; // зміна значення t через посилання ref
   cout<<"Остаточне  значення  t:  "  <<  t << endl;    //виводить 23
    return 0;
}
```

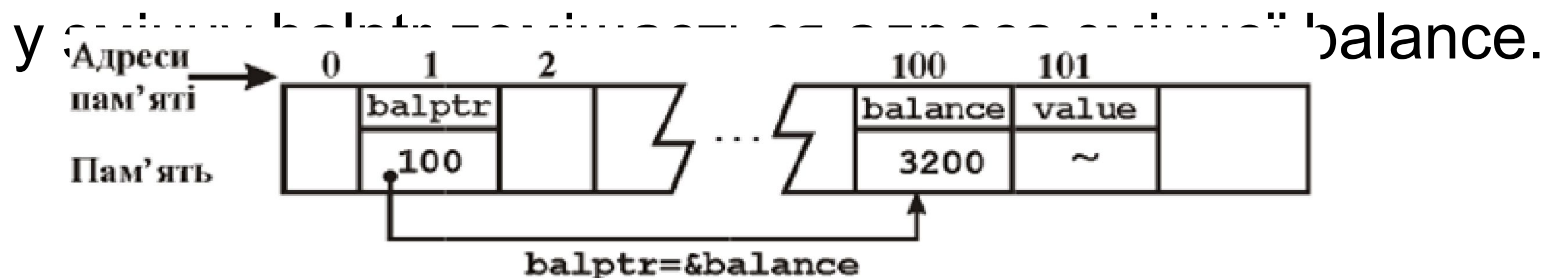
Вихід

Оператори, що використовуються з покажчиками

- З покажчиками використовуються два оператори: **"*"** і **"&".**
- **Операція взяття адреси змінної & повертає адресу свого операнда.** Оператор "&" – унарний.
- Наприклад:

balptr = &balance;

у [illegible] balance.



Оператори, що використовуються з покажчиками

Унарний оператор * звертається до значення змінної, розташованої за адресою, заданою його операндом.

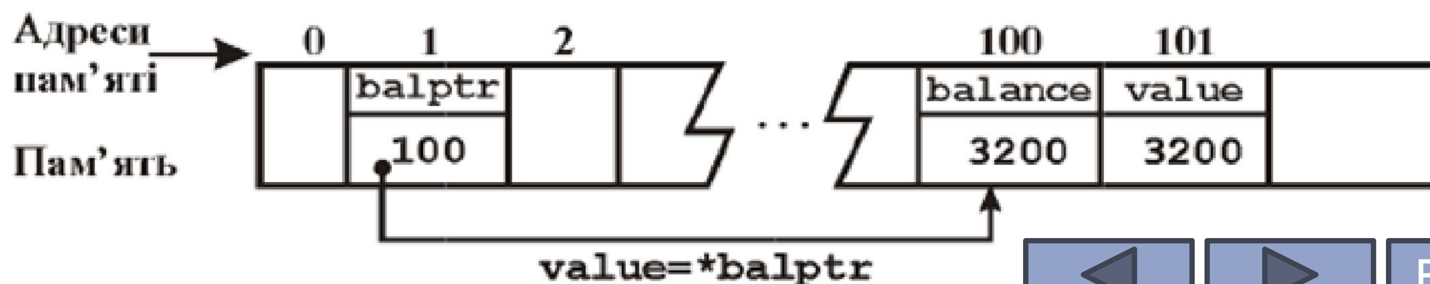
Даний оператор називають оператором **розіменування покажчика**.

Наприклад:

value = *balptr;

змінній value буде присвоєно значення змінної balance, на яку вказує змінна balptr.

Інструкцію можна прочитати так: "змінна value одержує значення (розташоване) за адресою balptr".



```
/* Програма 2: Оператори покажчиків */
#include <iostream>
using namespace std;
int main()
{
 int balance;
 int *balptr;
 int value;
 balance = 3200;
 balptr = &balance; //взяття адреси balance
 value = *balptr; //розіменування покажчика
 setlocale(LC_ALL, "ukr"); //для виведення кирилиці
 cout << "Баланс дорівнює:" << value <<'\n';
 return 0;
}
```

Баланс дорівнює: 3200

Оператори, що використовуються з покажчиками

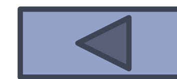
- **Оператори "*" і "&" мають більш високий пріоритет, ніж арифметичні оператори, за винятком унарного мінуса, пріоритет якого такий же,** як в операторів, що застосовуються для роботи з покажчиками.
- **Операції, виконувані за допомогою покажчиків, часто називають операціями непрямого доступу, оскільки ми побічно одержуємо доступ до змінної за допомогою деякої іншої змінної**.
- Операція непрямого доступу - це процес використання покажчика для доступу до деякого об'єкта.

Вихід

Оператори, що використовуються з покажчиками

- Покажчики можуть посилатися на інші покажчики.
- У комірках пам'яті, на які будуть посилатися перші покажчики, будуть міститися не значення, а адреси інших покажчиків.
- Число символів * при оголошенні покажчика показує порядок покажчика.
- Щоб отримати доступ до значення, на яке посилається покажчик його необхідно розіменовувати відповідну кількість разів.

```
/*Програма 3. Демонстрація покажчика на покажчики */
#include <iostream>
using namespace std;
int main() {
  int var = 123; // ініціалізація змінної var числом 123
  int *ptrvar = &var; // покажчик на змінну var
  int **ptr_ptrvar = &ptrvar;
// покажчик на покажчик на змінну var
  int ***ptr_ptr_ptrvar = &ptr_ptrvar;
  cout << " var\t\t= " << var << endl;
  cout << " *ptrvar\t= " << *ptrvar << endl;
  cout << " **ptr_ptrvar = " << **ptr_ptrvar << endl;
   // два рази розіменовуємо покажчик, через те, що він
другого порядку
```



Вихід

```
cout << " ***ptr_ptrvar = " << ***ptr_ptr_ptrvar << endl;
  // покажчик третього порядку
  cout << "\n ***ptr_ptr_ptrvar -> **ptr_ptrvar -> *ptrvar
-> var -> "<< var << endl;
  cout << "\t " << &ptr_ptr_ptrvar<< " -> " << " " <<
&ptr_ptrvar << " ->" << &ptrvar << " -> " << &var << " -> "
<< var << endl;
  return 0;
}
```

var = 123

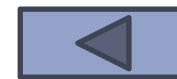
*ptrvar = 123

**ptr_ptrvar = 123

***ptr_ptrvar = 123

***ptr_ptr_ptrvar -> **ptr_ptrvar -> *ptrvar -> var -> 123

0xffffcc20 -> 0xffffcc28 ->0xffffcc30 -> 0xffffcc3c -> 123



3. Операції з покажчиками

- При присвоюванні значення області пам'яті, що адресується покажчиком, покажчик можна використовувати з лівої сторони від оператора присвоювання. Наприклад,

***p = 101;**

- Значення покажчика після операцій ++ чи -- збільшується або зменшується на довжину об'єкта, зв'язаного з покажчиком.

- Нехай Val(e) – значення виразу e. Тоді для покажчика p типу T:

Val(p++) = Val(p--) = Val(p), Val(++p) = Val(p)+sizeof(T),

Val(--p) = Val(p)-sizeof(T).

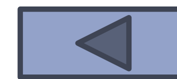
- Можна використовувати інструкцію, подібну наступній.

(*p)++;

- Круглі дужки тут обов'язкові, оскільки оператор "*" має більш низький пріоритет, ніж оператор "++".

Арифметичні операції над покажчиками

- З покажчиками можна використовувати тільки чотири арифметичних оператори: ++, --, + і -.
- **Коли покажчик інкрементується, він буде вказувати на область пам'яті, що містить наступний елемент базового типу цього покажчика.**
- **При кожному декрементуванні він буде вказувати на область пам'яті, що містить попередній елемент базового типу цього покажчика.**
- Для покажчиків на символьні значення результат операцій інкрементування й декрементування буде таким же, як при "нормальній" арифметиці, оскільки символи займають тільки один байт



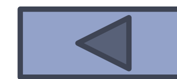
Вихід

Арифметичні операції над покажчиками

- У бінарних операціях додавання й віднімання можуть брати участь покажчик і ціле. Результатом операції буде покажчик того самого типу:

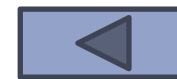
p±i=p±(i*sizeof(T)).

- В операції віднімання можуть брати участь два покажчики одного типу. Результат операції має цілий тип і дорівнює кількості об'єктів між зменшуваним і від’ємником.
- Результат буде від’ємним, якщо перша адреса менша ніж друга.



Порівняння покажчиків

- Два покажчики одного типу можна порівнювати в операціях ==, !=, <, <=, >, >=. Значення покажчиків розглядаються як цілі числа.
- Якщо, наприклад, p1 і p2 - покажчики, які посилаються на дві окремі й ніяк не зв'язані змінні, то будь-яке порівняння p1 і p2 у загальному випадку не має сенсу.
- Але **якщо p1 і p2 указують на змінні, між якими існує деякий зв'язок, наприклад, між елементами одного і того самого масиву, то результат порівняння покажчиків p1 і p2 може мати певний сенс.**



```
/* Програма 5. Адресна арифметика */
#include <iostream>
using namespace std;
int main(){
int i = 3;
int *p, *p1, *p2, v[20];
p1 = p = &v[4]; /*=&v[4]*/
cout << "p = " << p << endl;
cout << "p1 = " << p1 << endl;
(p++);  /*=&v[5]*/
cout << "p++ = " << p << endl;
(p--);   /*=&v[4]*/
cout << "p-- = " << p << endl;
(--p);   /*=&v[3]*/
cout << "--p = " << p << endl;
```

```
p2 = p1 + (i+4); /*=&v[11]*/
cout << "p2 = " << p2 << endl;
p1 = p2 - i;  /*=&v[8]*/
cout << "p1 = " << p << endl;
i = p1 - p2;  /*=-3*/
cout << "p1 - p2 = " << i << endl;
i= p2 - p1;  /*=3*/
cout << "p2 - p1 = " << i << endl;
if (p1 >= p2 )
  cout << "p1";
else
  cout << "p2";
 return 0;
}
```

Вихід

Покажчики й масиви

char str[80];

// str - масив, що містить 80 символів

char *p1;

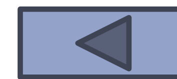
// p1 - покажчик на дані типу char

p1 = str;

/* адреса str[0] присвоюється покажчику p1, оскільки ім'я масиву без індексу str - це константний покажчик на початок цього масиву. Тепер str[4] або *(p1+4) – це п'ятий елемент масиву str */

У C++ **використання імені масиву без індексу генерує покажчик на перший елемент цього масиву**.

Ключовий момент, який необхідно чітко розуміти: **неіндексоване ім'я масиву, використане у виразі, означає покажчик на початок цього масиву**.



Вихід

Покажчики й масиви

- Одержати доступ до четвертого елемента масиву **str**, використовуйте один із наступних виразів:

- **str[3]**

або

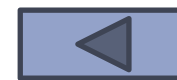
- ***(p1+3)**

Важливо! Переконаєтеся зайвий раз у правильності використання круглих дужок у виразі з покажчиками.

Вихід

Покажчики й масиви

- У C++ передбачено два способи доступу до елементів масивів: за допомогою індексування масивів і арифметики покажчиків.
- А**рифметичні операції над покажчиками іноді виконуються швидше, ніж індексування масивів, особливо при доступі до елементів, розташування яких відрізняється строгою впорядкованістю.**
- Використання покажчиків для доступу до елементів масиву – характерна риса багатьох C++-програм.
- **Іноді покажчики дозволяють написати більш компактний код у порівнянні з використанням індексування масивів.**



Вихід

```
/* Програма 6. Обчислення середнього значення додатних елементів масиву --- програма без використання покажчиків */
#include <iostream>
using namespace std;
int main( ){
    setlocale(LC_ALL, "ukr");
    int n, i, kilk = 0; ;
    cout << "Уведіть значення кількості елементів масиву:" << endl;
    cin >> n;
    float array[n], s = 0;
    cout << "Введення масиву:" << endl;
    for(i =0; i < n; i++)
        cin >> array[i];
```

```
for(i = 0; i < n; i++)
        if (array[i] > 0){
            s+=array[i]; //накопичення суми
            kilk++; //Підрахунок додатних елементів
            }
    cout.precision(4);
    cout << "Середн. арифм. додатніх елементів = " << s/kilk
<< endl;
    return 0;
}
```

Вихід

```
/* Програма 7. Обчислення середнього значення додатних
елементів масиву --- програма з використанням покажчиків */
#include <iostream>
using namespace std;
int main( )
{
    setlocale(LC_ALL, "ukr");
    int n, i, kilk = 0; ;
    cout << "Уведіть значення кількості елементів масиву:" <<
endl;
    cin >> n;
    float array[n], s = 0;
    cout << "Уведення масиву:" << endl;
```

```
    for (i = 0, s = 0; i < n; i++)
            {
                  cin >> *(array+i);
                  if (*(array+i) > 0)
                  {
                        s += *(array+i);
                        kilk++; }
      }
      cout.precision(4);
      cout << "Середн. арифм. додатніх елементів = " << s/kilk
<< endl;
        return 0;
}
```

Вихід

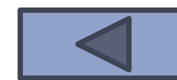
```
/* Програма 8. Використання арифметики покажчиків */
#include <iostream>
using namespace std;
int main ( )
{
    setlocale(LC_ALL, "ukr");
    int n, i, kilk = 0; ;
    cout << "Уведіть значення кількості елементів масиву:" <<
endl;
    cin >> n;
    float array[n], s(0);
    float * pm = array; //pm=&array[0];
```

```
    for (i = 0; i < n; i++) {
            cout << "Уведіть " << i << " елемент array" << endl;
            cin >> * pm++;
            cout << array[i] << endl;
            if (array[i] >0)
            {
              s+=array[i];
              kilk++; }
        }
        cout.precision(3);
         cout << "\n Середнє арифм. додатніх елементів = " <<
s/kilk << endl;
        return 0;
}
```

Вихід

Покажчики й масиви

- У цих програм може бути різна швидкодія, що обумовлено особливостями генерування коду С++-компіляторами.
- При **використанні індексування масивів генерується більш довгий код (з більшою кількістю машинних команд), ніж при виконанні арифметичних дій над покажчиками.**
- У професійно написаному С++-коді частіше зустрічаються версії, орієнтовані на обробку покажчиків.



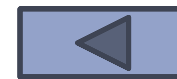
Вихід

Індексування покажчика

- У **С++ покажчик, що посилається на масив, можна індексувати так, ніби це було ім'я масиву** (це говорить про тісний зв'язок між покажчиками й масивами).
- Відповідно такому підходу синтаксис забезпечує альтернативу арифметичним операціям над покажчиками, оскільки він більш зручний у деяких ситуаціях.

Вихід

```
/* Програма 3. Індексування покажчика подібно
масиву */
#include <iostream>
#include <cctype>
using namespace std;
int main() {
 char str[40] = "We are learning the C++ language";
 char *p;
 int i;
 p = str; // str[5] = *(str+5) = p[5] = *(p+5)
 for(i=0; p[i]; i++)
    p[i] = toupper(p[i]); // Індексуємо покажчик
 cout << p; // Відображаємо рядок
 return 0;}
```

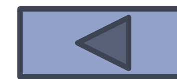


Вихід

Динамічний масив

- Для виділення динамічного масиву і роботи з ним використовуються окремі форми операторів new і delete: **new[]** і **delete[]**.
- Використання форми оператора delete для змінних при видаленні масиву призведе до таких несподіваних результатів, як пошкодження даних, витік пам’яті, збій або інші проблеми.
- При використанні new необхідно вказати тип і розмірність, наприклад:

```
int n = 100;
float *p = new float [n];
```



Динамічний масив

- Альтернативний спосіб створення динамічного масиву – використання функції malloc():

```
int n = 100;
float *q = (float *) malloc(n * sizeof(float));
```

Вихід

Масив покажчиків

Покажчики, подібно даним інших типів, можуть зберігатися в масивах.

Оператори описують масиви з покажчиками на об'єкти типу int та float:

```
int var;    /*звичайна ціла змінна*/
int array[10]; /*звичайний масив*/
int *ap[20];  /*масив покажчиків*/
int *pd[]={&array[0],&array[2],&array[4]}
/*ініціалізований масив покажчиків*/
ap[2]=&var; /*присвоює адресу цілої змінної var третьому елементу масиву ap*/
float *w[20]; /*масив дійсних покажчиків*/
```

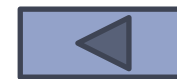
Вихід

```
/* Програма 10. Уведення і виведення масиву дійсних
покажчиків */
#include <iostream>
using namespace std;
int main ( )
{
    setlocale(LC_ALL, "ukr");
    int n, i, kilk = 0; ;
      cout << "Уведіть значення кількості елементів
масиву:" << endl;
    cin >> n;
```

Вихід

```
    float s(0), var;
        float *array = new float[n];
        for(i = 0; i < n; i++)
            cin >> array[i];

        for(i = 0; i < n; i++)
            cout << array[i] << " ";
        return 0;
}
```



Вихід

Масив покажчиків

- Подібно іншим масивам, масиви покажчиків можна ініціалізувати. Як правило, ініціалізовані масиви покажчиків використовуються для зберігання покажчиків на рядки. Наприклад, для масиву fortunes:

```
char *fortunes[] = {
 "Рядок1\n",
 "Рядок2\n",
 "Рядок3\n",
 "Рядок4\n",
 "Рядок5\n"
};
```

Покажчики й рядкові літерали

- Ім’я масиву символів є покажчиком-константою. Якщо рядок описано як масив символів, тоді можна змінювати символи рядка з допомогою покажчика.
- **Приклад.** Покажчик на масив символів. Доступ до символу рядка, який описаний як масив символів.

```
// покажчик на масив символів
 char str1[] = "Text"; // масив str1 = { 'T', 'e', 'x', 't', '\0' }
 char *p1;

// спосіб 1 - ім'я рядка є покажчиком на перший символ рядка
p1 = str1; // покажчик p1 вказує на str1
// спосіб 2
p1 = &str1[0];

//доступ до символу рядка через покажчик
*p1 = 'N';     // str1 = "Next" - працює
*(p1+3) = '!'; // str1 = "Nex!"
```

Покажчики й рядкові літерали

Читання символу в рядку символів з допомогою покажчика.

```
// читання символу рядкового літералу з допомогою покажчика

char * p1;

char c;

p1 = "Hello world!";


c = *p1;     // c = 'H'

c = *(p1+1); // c = 'e'

c = *(p1+2); // c = 'l'

c = *(p1+3); // c = 'l'
```

Вихід

Покажчики й рядкові літерали

/* читання символу масиву символів з допомогою покажчика*/

```
char *str = "Hello world!";
char *p2 = str;


c = *p2;       // c = 'H'
c = *(p2+1);  // c = 'e'
c = *(p2+12); // c = '\0'
```

Вихід

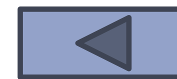
Покажчики й рядкові літерали

- **Таблиця рядків** - це таблиця, згенерована компілятором для зберігання рядків, використовуваних у програмі.
- С++-компілятори створюють оптимізовані таблиці, у яких один рядковий літерал може використовуватися у двох (або більш) різних місцях програми. Тому "насильницька" зміна рядка може викликати небажані побічні ефекти.
- ***Рядкові літерали являють собою константи***, і деякі сучасні С++-компілятори попросту не дозволять міняти їхній вміст.

Вихід

```
/* Програма 11. Версія з адресною арифметикою */
#include <iostream>
#include <cstdio>
#include <windows.h>

 using namespace std;
int main() {
 char str[80], token[80];
 char *p, *q; // Покажчики
 SetConsoleOutputCP(1251);
 SetConsoleCP(1251);
 cout << "Уведіть речення: ";
 gets(str);
 p = str;
```



Вихід

```
while(*p) {
  // Читаємо слово до ' ' або '\0'
   q = token;
   while(*p != ' ' && *p) {
    *q = *p;
    q++; p++;
   }
   // Рух за пробіл
   if(*p) p++;
   // Завершуємо слово ‘\0’
   *q = '\0';
   cout << token << '\n';
  }
  return 0;
}
```

Вихід

```
/* Програма 12. Версія з індексацією масива */
#include <iostream>
#include <cstdio>
#include <windows.h> //для виведення кирилиці
using namespace std;

int main() {
 char str[80], token[80];
 int i, j; // Індексні змінні
 SetConsoleCP(1251); //для виведення кирилиці
 SetConsoleOutputCP(1251); //для виведення кирилиці
 cout << "Уведіть речення: ";
 gets(str);
```

Вихід

```
for(i=0; ; i++) {
 // Читаєм слово до ' ' або '\0'
  for( j=0;
     str[i]!=' ' && str[i];
     j++, i++ )
   token[j] = str[i];
  /* Вже за пробілом або в кінці рядка
   Завершуємо слово '\0' */
  token[j] = '\0';
  cout << token << '\n';
  if(!str[i]) break;
 }
 return 0;
}
```

ЗАПИТАННЯ

?