



*ОРГАНІЗАЦІЯ ДАНИХ ТА АЛГОРИТМИ ЇХ ОПРАЦЮВАННЯ.
ВКАЗІВНИКИ ТА ПОСИЛАННЯ.
СТАТИЧНЕ І ДИНАМІЧНЕ ВИДІЛЕННЯ ПАМ'ЯТІ*

Посилання (reference) – це нововведення C++ (їх немає в C). *Посилання* є альтернативним ім'ям (синонімом) змінної і для нього *місце в оперативній пам'яті не резервується*. Посилання дозволяють набагато зручніше передавати інформацію у функцію за допомогою адрес, аніж вказівники.

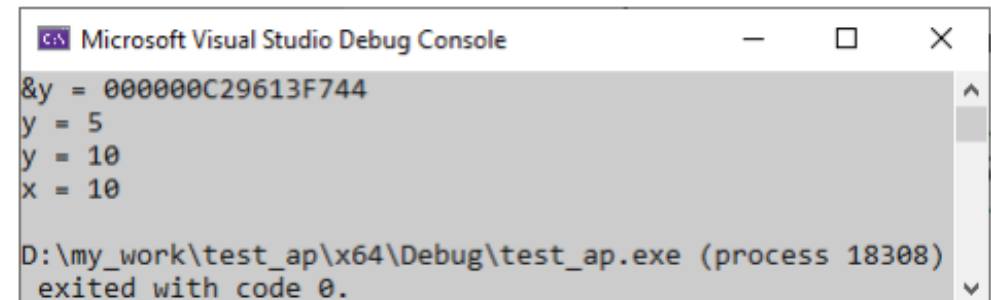
Оголошення посилань

```
<тип_даних> & <ім'я_посилання> = <ім'я_змінної>;
```

```
int x;  
int& y = x;
```

*слід ініціалізувати посилання
в момент його оголошення*

```
// оголошення та визначення посилання на константу  
const double& e = 2.72;  
// оголошення та визначення посилання на змінну  
int x = 5;  
int& y = x;  
cout << "&y = " << &y << endl; // результат: 000000C29613F744  
cout << "y = " << y << endl;   // результат: 5  
//зміна значення  
y = 10;  
cout << "y = " << y << endl;   // результат: 10  
cout << "x = " << x << endl;   // результат: 10
```



```
Microsoft Visual Studio Debug Console  
&y = 000000C29613F744  
y = 5  
y = 10  
x = 10  
  
D:\my_work\test_ap\x64\Debug\test_ap.exe (process 18308)  
exited with code 0.
```

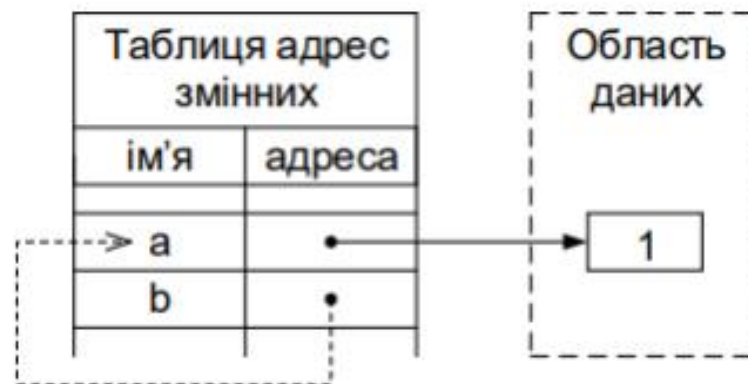
Розподіл пам'яті для посилань

```
int a = 1; int& b = a;
```

Розглянемо детальніше розподіл пам'яті: створюється таблиця адрес змінних, в яку заноситься ім'я змінної та адреса тої комірки, яка виділена для зберігання значення цієї змінної:



Для посилань – зберігається не адреса комірки з області даних, а адреса відповідного елемента таблиці адрес змінних:

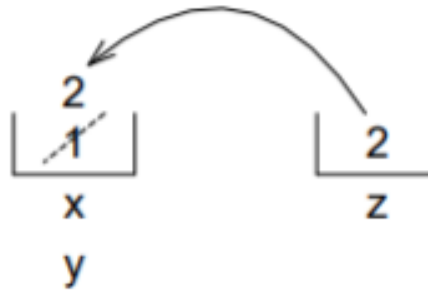


Типові помилки при оголошенні та використанні посилань

- 1) посилання іншого типу, аніж змінна, на яку воно посилається.
- 2) не присвоїти початкове значення змінній-посиланню (це значення має бути іменем іншої змінної, яка вже відома на момент оголошення посилання).

- 3) Спроба «переналаштувати» посилання.

```
int x = 1;
int& y = x; // посилання на x
int z = 2;
y = z; // "переналаштовуємо" посилання
```

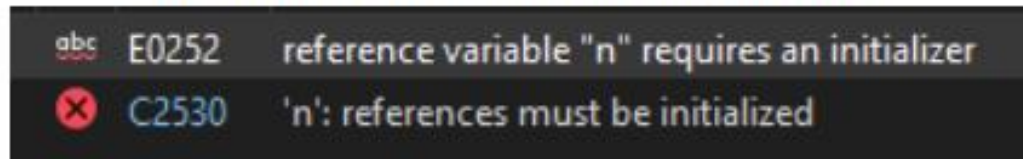


```
int& n = m; // вірно
int &n = m; // рівносильна команда
```

- 4) повернення із функції вказівника чи посилання на локальну автоматичну змінну.
- 5) невірне тлумачення команди оголошення кількох змінних та посилань.

```
int& n, m;
```

оголошене посилання не ініціалізоване.



Обмеження на оголошення посилань:

- 1) Не можна визначати вказівники на посилання;
- 2) Не можна створювати масиви посилань;
- 3) Не можна оголошувати посилання на посилання;

Вказівники

Програміст може визначати власні змінні для зберігання адрес. Такі змінні, які містять значення адрес, називаються вказівниками.

Вказівник на змінну (pointer) – це місце розташування змінної в пам'яті комп'ютера, тобто її адреса.

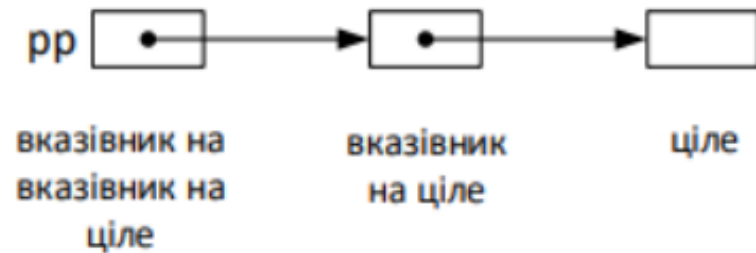
В мові C++ є *три види вказівників*:

- вказівник на об'єкт;
- вказівник на функцію;
- вказівник на void.

Тип даних може бути будь-яким, крім посилання чи бітового поля

Оголошення вказівників

`<тип_даних> *<ім'я_вказівника> = [<значення>];`



```
int *p; // оголошення вказівника
```

```
int* p; // рівнозначно
```

```
int** pp; // оголошення вказівника на вказівник
```

Вказівник може бути константним або змінним об'єктом, і може вказувати на константний чи змінний об'єкт.

1) `const` – праворуч від `*`

`<тип> * const <вказівник>`



Тоді вказівник – це константний об'єкт; його зміна – заборонена. (не можна змінювати адресу об'єкта, на який налаштований вказівник).

1) `const` – ліворуч від `*`

`<тип> const * <вказівник>`



Тоді це – вказівник на константний об'єкт. (не можна змінювати значення об'єкта, на який налаштований цей вказівник).

Операція & – отримання адреси (отримання вказівника)

Операція & дозволяє отримати адресу деякого об'єкта в оперативній пам'яті. Результатом операції буде адреса у деякому внутрішньому представленні, тобто – вказівник на цей об'єкт. & – унарна операція, її знак записується перед операндом:

```
int i;  
int* pi = &i; //вказівнику pi присвоїли значення адреси змінної i
```

Операцію & можна застосовувати лише до величин, які мають ім'я та розміщуються в оперативній пам'яті.

Ініціалізація вказівників

1) Присвоєння вказівнику адреси вже створеного об'єкта

- За допомогою операції отримання адреси

```
int a = 5; // ціла змінна  
int* p = &a; // вказівник p отримує значення адреси змінної a
```

- За допомогою значення іншого, вже ініціалізованого вказівника

```
int* r = p; // r отримує значення адреси, яка зберігається у  
           вказівнику p
```

2) Присвоєння вказівнику адреси області пам'яті в явному вигляді

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

(char*) – операція приведення типу, ціла шістнадцяткова константа перетворюється до типу char* – «вказівник на символ»

3) Присвоєння вказівнику нульового значення

```
int* a = NULL;  → nullptr  
int* b = 0;
```

Ініціалізація вказівників

Найчастіше вказівники використовують при роботі з *динамічною пам'яттю*— область оперативної пам'яті, виділенням і звільненням якої керує програміст. Змінні, які розміщуються в динамічній пам'яті називаються *динамічними*.

Доступ до динамічних змінних можливий лише за допомогою вказівників на них.

Динамічні змінні – це змінні, створенням та знищенням яких керує явними командами сам програміст, вони створюються та знищуються окремими командами під час виконання програми.

Час існування динамічних змінних – від моменту їх створення до завершення виконання програми або до їх явного знищення.

В мові C++ є два способи роботи з динамічною пам'яттю:

- 1) успадкований від мови C: використовує набір функцій malloc() (memory allocation) та free();
- 2) більш зручний, реалізований в мові C++: використовує операції new та delete.

4) Виділення області динамічної пам'яті та присвоєння вказівнику адреси початку цієї області

- За допомогою операції **new**

```
int* n = new int; // 1  
int* m = new int(10); // 2  
int* q = new int[10]; // 3
```

Якщо пам'ять була виділена операцією **new**, то її слід звільняти за допомогою операції **delete**.

```
delete n; // 1  
delete m; // 2  
delete[] q; // 3
```

засмічення пам'яті.

Якщо змінна-вказівник виходить за межі області видимості (тобто, за межі блоку, в якому вона визначена), то пам'ять виділена для вказівника – звільниться (як це відбувається і для будь-якої локальної змінної). При цьому пам'ять, виділена для динамічної змінної, – звільнена не буде.

Якщо вказівник, який був налаштований на певну динамічну змінну, отримує інше значення, то попереднє значення втрачається.

Операції з вказівниками



Операція розіменування (розадресації) – використовується для доступу до області пам'яті, адреса якої міститься у вказівнику:

```
char a;  
char* p = new char;  
*p = 'A';  
a = *p;
```



Арифметичні операції

`+=` збільшення на константу;

`-=` зменшення на константу;

`-` різниця;

`++` інкремент;

`--` декремент;

Операція присвоєння

Якщо у виразі використовуються вказівники різних типів, то необхідне явне приведення типів для всіх вказівників, крім `void*`. Присвоєння без явного приведення типу допускається лише у двох випадках:

1) вказівникам типу `void*`:

```
char* p = new char;  
*p = 'A';  
void* v = p; // присвоєння без явного приведення типу
```

2) якщо типи вказівників (який присвоюється і якому присвоюється, тобто, записаних як після, так і перед знаком операції присвоєння) – однакові.

Зауважимо, що операція додавання двох вказівників не допускається:

```
int a = 2, b = 3, i, * ptr1 = &a, * ptr2 = &b;  
s = ptr1 + ptr2;    // ПОМИЛКА! заборонена операція
```

При записі виразів із вказівниками особливу увагу слід звертати на пріоритети операцій.

```
int x = 3, * p = &x;  
*p++ = 8; // еквівалентно послідовності команд 1)*p=8; 2)p++; x=8
```

Операції розадресація та інкремент мають однаковий пріоритет і виконуються справа наліво. Але, інкремент – постфіксний, тому він виконується після виконання операції присвоєння. Таким чином, спочатку за адресою, записаною у вказівнику *p*, буде записано значення 8, а потім вказівник буде збільшений на кількість байт, що відповідає його типу.

Вираз $(*p)++$, навпаки, інкрементує значення, на яке вказує вказівник:

```
p = &x;  
(*p)++;    // результат: x=9
```

При порівнянні вказівників порівнюються адреси, що зберігаються у вказівниках

Результатом порівняння вказівників є false (0) або true (1).

Дякую за увагу

Лектор:
кандидат фіз.-мат. наук, доцент
Шаклеїна Ірина
iryna.o.shakleina@lpnu.ua
кафедра ІСМ, ІКНІ