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

Посилання

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.

Посилання можна використовувати як псевдоніми для інших змінних (не лише в якості параметрів функцій), – правда, для цього є мало підстав і причин.

Оголошення посилань

Оголошення посилання має вигляд:

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

Наприклад:

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

Обов’язково *слід ініціалізувати посилання в момент його оголошення*. Типи посилання та змінної, значенням якої ініціалізовується посилання, мають співпадати, інакше створиться анонімний об'єкт для змінної, що не була оголошена. Після ініціалізації посилання завжди посилається на певний об'єкт. Під час виконання програми *не можна змінити посилання* з одного об'єкта на інший.

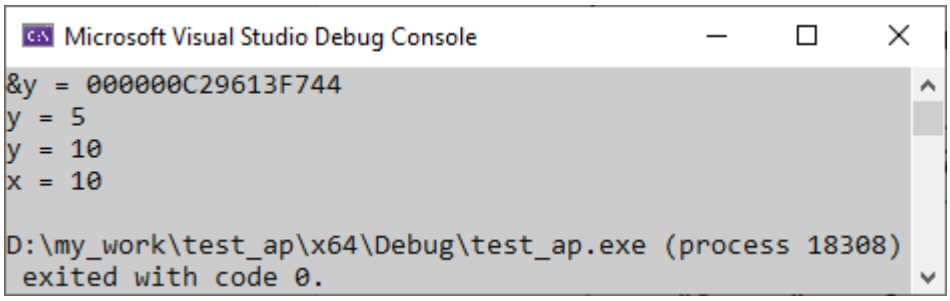
Параметр-посилання ініціалізується в момент виклику функції, а змінну-посилання слід ініціалізувати в момент її визначення.

Використання посилання після його ініціалізації дає той же результат, що й безпосереднє використання змінної.

Наприклад:

```
// 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
```



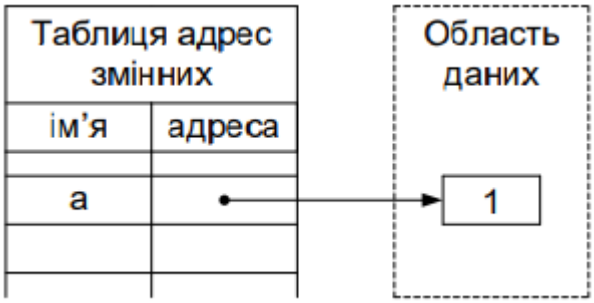
Посилання не може існувати незалежно від певної змінної.

Розподіл пам'яті для посилань

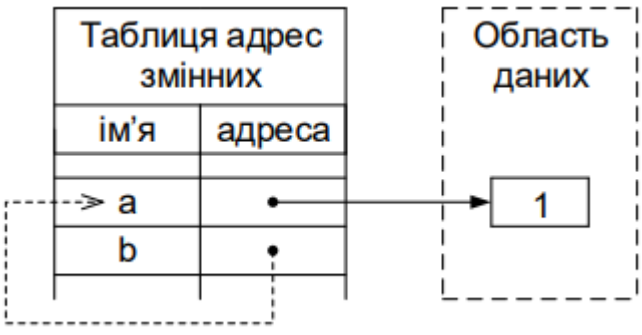
Розглянемо приклад:

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

Розглянемо детальніше розподіл памяті: 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:



Тому при оголошенні посилання його слід обов'язково ініціалізувати, тобто, – вказати ім'я змінної, для якої це посилання стане синонімом. При цьому встановлюється зв'язок в таблиці адрес змінних між посиланням та відповідним елементом таблиці (змінній, на яку налаштовується посилання).

Типові помилки при оголошенні та використанні посилань

1) *посилання іншого типу, аніж змінна, на яку воно посилається*. В C++ є

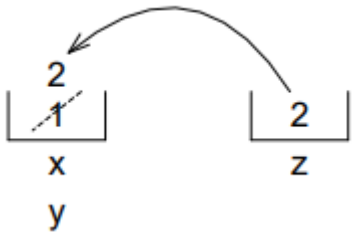
правило: *Посилання має бути того самого типу, що і змінна, на яку воно посилається.*

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. Наприклад:

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

Розглянемо детальніше наведений фрагмент коду. В дійсності відбувається копіювання значень: значення змінної z копіюється в комірку з іменем y (а оскільки y – це синонім для x, то значення змінної z копіюється в комірку, виділену для x). В результаті виконання команд всі вказані змінні будуть мати значення 2.



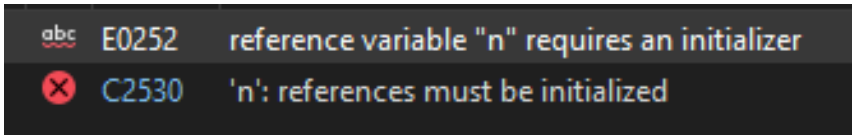
4) *повернення із функції вказівника чи посилання на локальну автоматичну змінну*.

Після виходу із області дії (тобто, після завершення виконання функції) локальні змінні перестають існувати. Пам’ять, виділена для локальних змінних, – звільняється, а їх імена – стають невідомим (локальні змінні автоматично знищуються). Спроба повернути вказівник чи посилання на локальну змінну означає намагання повернути адресу чи псевдонім вже неіснуючого в пам’яті об’єкта.

5) *невірне тлумачення команди оголошення кількох змінних та посилань*.

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

– в даному прикладі оголошене посилання не ініціалізоане.



Немає значення, де записувати символ & «амперсенд» при визначенні посилання – біля типу, чи біля змінної:

```
int& n= m; // вірно
 int &n = m; // також вірно, це рівносильна команда
```

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 С++ є *три види вказівників*:

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

Вони відрізняються властивостями та допустимими операціями. *Вказівники* – це не самостійний тип даних, він *завжди пов’язаний з якимось іншим конкретним типом*.

Робота з вказівниками позбавляє від необхідності переміщення об'єктів у пам'яті, яке займає багато часу, якщо об'єкти є великими. При використанні вказівників змінюються вказівники на адреси, а не на дані.

Оголошення вказівників

Синтаксис оголошення вказівника такий:

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

де тип_даних – простий або структурований тип адресованої змінної; символ * означає “вказати на”.

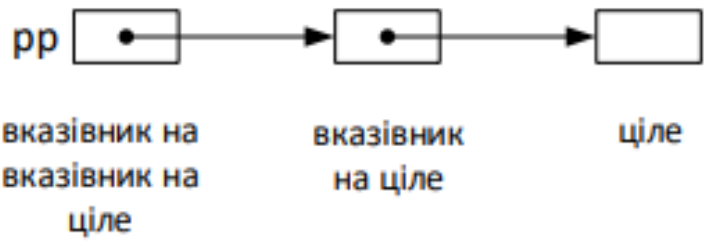
Вказівник на об’єкт

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

Тип даних може бути будь-яким, крім посилання чи бітового поля (тобто, не можна оголошувати вказівники на посилання чи вказівники на бітові поля);

Можна оголосити вказівник на вказівник тощо:

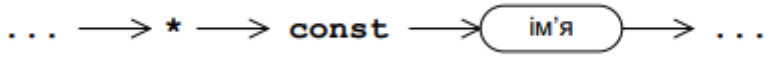
```
int** pp; // оголошення вказівника на вказівник
```



Вказівник може бути константним або змінним об'єктом, і може вказувати на константний чи змінний об'єкт. Можна виділити наступні випадки:

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

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 С++ є два способи роботи з динамічною пам'яттю:

1) успадкований від мови С: використовує набір функцій malloc() (memory allocation) та free();

2) більш зручний, реалізований в мові С++: використовує операції new та delete.

При визначенні вказівника рекомендується виконати його ініціалізацію. Використання не ініціалізованих вказівників – типова помилка!

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 (NULL)

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

– в першому випадку використовується константа `NULL`, визначена як 0. Таким чином, обидва зазначених способи – еквівалентні. Вказівник `NULL` (0) реалізовано як адресу 0, при цьому гарантується, що об'єктів з адресою 0 немає і ніколи не буде. Тому це значення `NULL` (0) можна використовувати для перевірки, чи налаштований наш вказівник на конкретний об'єкт, чи ні.

Рекомендується використовувати значення 0, а не `NULL`, бо в деяких реалізаціях `NULL` може бути не визначеним, в інших – використання `NULL` інколи може приводити до помилки. В стандарті С++11 для позначення нульового вказівника добавлене ключове слово `nullptr.`

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
```

Команда 1 створює динамічну змінну цілого типу і налаштовує вказівник n на цю змінну. Тобто, виділяється область динамічної пам'яті, достатня для розміщення величини типу `int`, і адреса початку цієї області записується у змінну n.

Команда 2 створює динамічну змінну цілого типу, налаштовує вказівник m на цю змінну та ініціалізує її значенням 10. Зауважимо, що ініціалізатор в цьому випадку має вигляд (10), а не = 10, оскільки команда з ініціалізатором = 10 – недопустима (вона приводить до помилки при компіляції).

Команда 3 створює динамічний масив із 10 елементів цілого типу і налаштовує вказівник q на цей масив. Тобто, виділяється пам'ять для 10 величин типу `int` (динамічного масиву із 10 елементів), і записує адресу початку цієї області у змінну q, яка може трактуватися як ім'я масиву.

Якщо пам'ять була виділена операцією `new`, то її слід звільняти за допомогою операції `delete`. Створені в попередніх прикладах динамічні змінні слід знищувати наступними командами :

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

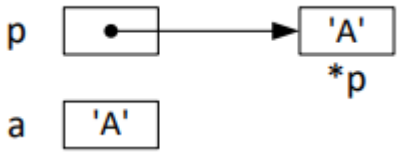
Якщо змінна-вказівник виходить за межі області видимості (тобто, за межі блоку, в якому вона визначена), то пам'ять виділена для вказівника – звільниться (як це відбувається і для будь-якої локальної змінної). При цьому пам'ять, виділена для динамічної змінної, – звільнена не буде. Оскільки знищується вказівник на динамічну змінну, то ця динамічна змінна стає не доступною. Це –причина засмічення пам'яті.

Якщо вказівник, який був налаштований на певну динамічну змінну, отримує інше значення (наприклад, в результаті присвоєння), то попереднє значення безслідно втрачається. Така динамічна змінна стає не доступно. Це –причина засмічення пам'яті

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;
```



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.

Значення 0 неявно перетворюється до вказівника на будь-який потрібний в конкретному виразі тип. Не можна присвоювати значення вказівникам-константам – тобто, не можна змінювати значення константних вказівників (як і констант будь-якого типу). Можна присвоювати значення вказівникам на константи чи змінним, на які налаштовані константні вказівники

Арифметичні операції

Є наступні арифметичні операції із вказівниками:

+= збільшення на константу;

-= зменшення на константу;

- різниця;

++ інкремент;

-- декремент;

Ці операції автоматично враховують розмір типу тих величин, які адресуються відповідними вказівниками. Ці операції можна застосовувати лише до вказівників одного і того самого типу, вони мають зміст в основному для структур даних, які послідовно розміщені в оперативній пам'яті, – наприклад, для масивів.

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).