Лекція 9. Вказівники. Операції над вказівниками

Вказівники

Оперативна пам'ять складається з послідовний комірок. Кожна комірка має *номер*, називаний **адресою.**

В 32-бітних системах можна адресувати 2^{32} байт (4Гб) пам'яті, в 64-бітних – 2^{64} відповідно.

Змінна (або константа), що зберігає адресу, називається **вказівником.**

Для чого потрібні вказівники

Вказівники підвищують ***гнучкість доступу до даних***:

1. Замість самих даних можна зберігати вказівник на них. Це дозволяє зберігати дані в одному екземплярі й множина вказівників на ці дані. Через різні вказівники ці дані можна обновляти (приклад – корпоративна БД).
2. Вказівнику можна привласнити адреса іншого об'єкта (замість старого з'явився новий телефонний довідник).
3. За допомогою вказівників можна створювати складні **структури даних.**

Типи вказівників

Вказівники діляться на:

- ***Типізовані*** (указують на об'єкт деякого типу)
- Мають тип: **^<тип>**
- Приклад. `^integer` – вказівник на integer
- ***Безтипові*** (зберігають адреса комірки пам'яті невідомого типу)
- *Перевага:* можуть зберігати що завгодно
- Мають тип: **`pointer`**

Приклад коду.

```
var
  i: integer := 5;
  r: real := 6.14;

  pi: ^integer;
  pr: ^real;

begin
  pi := @i;
  pr := @r;
  pi := @r; // ПОМИЛКА компіляції
end.
```

@ – унарна операція **взяття адреси**

Операція розадресації (розіменування)

```
var
  i: integer := 5;
  pi: ^integer;

begin
  pi := @i;

  pi^ := 8 - pi^;
  writeln(i); // 3
end.
```

^ – операція **розіменування**

pi^ – те, на що вказує pi, тобто інше ім'я i або **посилання** на i.

Отут треба згадати визначення посилання:

Посилання – інше ім'я об'єкта.

Нульовий вказівник

Всі глобальні *неініціалізовані* вказівники зберігають спеціальне значення **nil**, що говорить про те, що вони ***нікуди не вказують***.

Вказівник, що зберігає значення **nil** називається **нульовим.**

```
var
  pi: ^integer; //вказівник pi зберігає значення nil
  i: integer;

begin
  pi := @i;     //pi зберігає адреса змінної i
  pi := nil;    //pi знову нікуди не вказує

  pi^ := 7;     //ПОМИЛКА часу виконання:
                //спроба розіменовувати нульовий вказівник
```

Спроба розіменувати нульовий вказівник приводить до ***помилки часу виконання***.

Безтипові вказівники

```
var
  p: pointer;
  i: integer;

begin
  p := @i;
end.
```

Безтиповому вказівнику можна привласнити адреса змінної будь-якого типу, тобто безтиповий вказівник *сполучимо по присвоюванню з будь-яким типовим* вказівником.

Спроба розіменовувати безтиповой вказівник приводить до ***помилки компіляції***. Тобто він може тільки *зберігати* адреси.

Виявляється, будь-який типізований вказівник *сполучимо по присвоюванню з безтиповим*, тобто наступний код вірний:

```
var
  pi: ^integer;
  i: integer;
  p: pointer;

begin
  p := @i;
  pi := p;
  pi^ += 2;
end.
```

Питання. Чи не можна інтерпретувати пам'ять, на яку вказує p, що як належить до певного типу?

Відповідь – так, можна. От як це зробити:

```
type
  pinteger = ^integer;
var
  i, j: integer;
  p: pointer;
```

```
begin
  p := @i;
  pinteger(p)^ := 7; //використовуємо явне приведення типу
  writeln(i); // 7
end.
```

Запис

<тип>(<змінна>)

показує, що використовується **явне приведення типів**.

Увага! Неконтрольована помилка!

```
type
  preal = ^real;
var
  i, j: integer;
  p: pointer;

begin
  p := @i;
  preal(p)^ := 3.14; //ПОМИЛКА!
end.
```

Область пам'яті, на яку вказує p трактується як область, що зберігає речовинне число (8 байт), і тому константа 3.14 записується в ці 8 байт. Однак, змінна i займає тільки 4 байти, тому затираються ще 4 сусідніх байти (у цьому випадку вони належать змінної j).

Доступ до пам'яті, що має інше внутрішнє представлення

```
type
  Rec = record
    b1, b2, b3, b4, b5, b6, b7, b8: byte;
  end;

  PRecord = ^Rec;

var
  r: real := 3.1415;
  prec: ^Rec;

begin
  var temp : pointer := @r;
  prec := temp;
  writeln(prec^.b1, ' ', prec^.b2, ' ', {..., } prec^.b8);
end.
```

Зауваження. Важливо, що типи real і Rec мають один розмір.

Неявні вказівники в язиці Pascal

1. `procedure p(var i: integer)`
2. Для параметра-змінної при виклику на стек кладе не сама змінна, а вказівник на неї.
3. `var pp: procedure(i: integer)`
4. Для зберігання процедурної змінної використовується комірка пам'яті, що є вказівником.
5. **var** a: **array of** real;
6. Змінна типу динамічний масив є вказівником на дані масиву, що зберігаються в динамічній пам'яті.

Динамічна пам'ять

Особливості динамічної пам'яті

Пам'ять, що належить програмі, ділиться на:

- **Статичну** (пам'ять, займана глобальними змінними й константами).
- **Автоматичну** (пам'ять, займана локальними даними, тобто стік програми).
- **Динамічну** (пам'ять, виділювана програмі по спеціальному запиті).

На додаток до статичної й автоматичної пам'яті, які фіксовані після запуску програми, програма може одержувати нефіксовану кількість *динамічної* пам'яті. Обмеження на обсяг виділюваної динамічної пам'яті зв'язані лише з настроюваннями операційної системи й обсягом оперативної пам'яті комп'ютера.

Основна проблема - явно виділену динамічну пам'ять необхідно повертати, інакше не вистачить пам'яті іншим програмам.

Для явного *виділення* й *звільнення* динамічної пам'яті використовуються процедури:

- **New**
- **Dispose**

```
var
  p: pinteger;  //p нікуди не вказує
begin
  New(p);       //у динамічній пам'яті виділяється комірка
                //розміром під один integer, і
                //p починає вказувати на цю комірку
  p^ := 3;
  Dispose(p);   //повертає динамічну пам'ять,
                //контрольовану вказівником p, назад ОС
end.
```

По закінченні роботи програми, вся викликана програмою динамічна пам'ять вертається ОС.

Але краще звільняти динамічну пам'ять явно! Інакше в процесі роботи програми вона може займати більші обсяги (ще не звільненої) пам'яті, що шкодить загальної продуктивності системи.

Помилки при роботі з динамічною пам'яттю

1.

```
var p: pinteger;
begin
  p^ := 5;  //ПОМИЛКА
end.
```

Помилка ***розіменування нульового вказівника*** (спроба використовувати невиділену динамічну пам'ять).

2.

```
var p: pinteger;
begin
  New(p);
  New(p);  //ПОМИЛКА
end.
```

Витік пам'яті (пам'ять, що виділилася в результаті першого виклику `New(p)`, належить програмі, але не контролюється ніяким вказівником.

2a.

```
procedure q;
var p: pinteger;
begin
```

```
  New(p);
end;
begin
  q;  //ПОМИЛКА
end.
```

Витік пам'яті в підпрограмі: звичайно якщо динамічна пам'ять виділяється в підпрограмі, то вона повинна в цій же підпрограмі вертатися. Виключення становлять т.зв. "створюючі" п/п:

```
function CreateInteger: pinteger;
begin
  New(Result);
end;

begin
  var p: pinteger := CreateInteger;
  p^ := 555;
  Dispose(p);
end.
```

Відповідальність за видалення пам'яті, виділеної в підпрограмі, лежить на програмісті, що викликав цю підпрограму.

3.

```
var p: pinteger;
begin
  for var  i:=1 to 1000000 do
    New(p);  //ПОМИЛКА
end.
```

Out of Memory (дуже більші витоки пам'яті, у результаті яких динамічна пам'ять може «вичерпатися»).

4.

```
var p: pinteger;
begin
  New(p);
  p^ := 5;
  Dispose(p);
  p^ := 7;  //ПОМИЛКА
end.
```

Після виклику `Dispose(p)`, `p` називають ***висячим вказівником*** (тому що він указує на недоступну більше область пам'яті).