1.1. МАСИВИ

Наведемо ще раз визначення масиву.

Масив – це упорядкована послідовність однотипних елементів, кожен з яких має один і той же тип. Тип елементів називається *базовим* і може бути будь-яким, навіть структурованим, наприклад, запис (record), який має заздалегідь визначений об'єм пам'яті. Масиви зберігаються в оперативній пам'яті комп'ютера. Елементи масиву розміщені впорядковано, кожен має свій номер, який має назву індекс. Доступ до елементів масиву відбувається шляхом вказування імені масиву та його порядкового номера у послідовності, тобто використовується прямий метод доступу. Індексом може бути будь-який арифметичний вираз ординального типу. Одномірний масив можна розглядати як вектор, а функція адресації примірників його елементів показано на рис. 1.1.

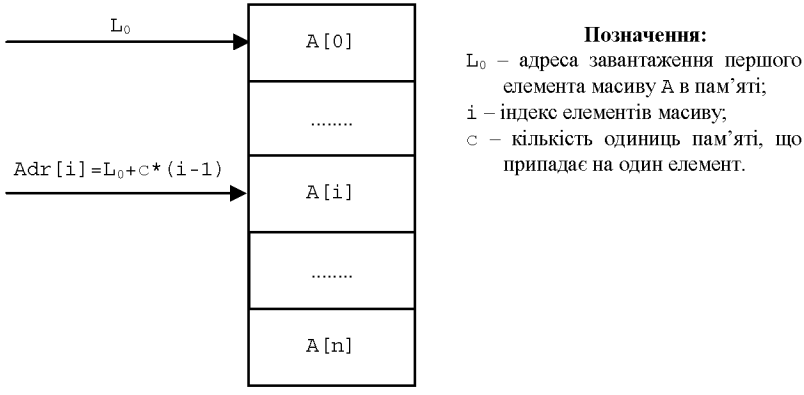

Рис. 1.1. Адресація елементів одномірного масиву

У зв'язку з тим, що елементом масиву може бути будь-яка статична структура, то можливо створення масиву масивів – двомірного масиву і т. д., тобто масиви можуть бути не тільки одномірним, але й n-мірними. При n=2 такий масив можна інтерпретувати як таблицю, при n=3

як куб, а в загальному випадку, n-мірний гіперкуб. Але й багатомірний масив в пам'яті ЕОМ реалізується як лінійна структура з лінійною адресацією. Це здійснюється за рахунок того, що будь-який багатомірний масив може бути представлений еквівалентним одномірним. Так, для двомірного масиву `A[m,n]`, що має індексацію `A[i,j]`, функція адресації елементів масиву має такий вигляд:

$$Adr[i,j]) = L_0 + (j-1)*n+i-1,$$

де: $1 \le i \le n$; $1 \le j \le n$.

Масиви за своєю організацією можуть бути *статичними:* кількість елементів, а отже і розмір виділеної пам'яті, для них визначається під час компіляції програми, а також *динамічними:* кількість елементів яких визначається і може бути змінена як в одну, так і в іншу сторону під час виконання програми. Останній варіант є більш практичним за рахунок суттєвої економії оперативної пам'яті при обробці заздалегідь невизначеної довжини послідовності.

У прикладах 1.1 та 1.2 наведено опис, ініціалізацію та використання статичного та динамічного масивів (тут і далі мовою Pascal).

1.1. Приклад опису та ініціалізації статичного масиву

```
type
   Person=record
     Name: string[15];
     YearOfBirth: integer;
   end;
   Mas1=array[1..3] of Person;
const
   M: Mas1=(
      (Name: 'Ivanov'; Age: 45),
      (Name: 'Петров'; Age: 55),
      (Name: 'Шевченко'; Age: 35));
var
   P: Person;
begin
   P:=M[2]; //наприклад...
   ..................
```

1.2. Приклад опису та ініціалізації динамічного масиву

```
type
   Person=record
     Name: string[15];
```

```
    YearOfBirth: integer;
  end;
  Mas1=array of Person;
var
  P: Person;
  M: Mas1;
begin
  SetLength(M,3);
  Mas1[0].Name:='Ivanov';
  Mas1[0].Age:=45;

  Mas1[1].Name:='Петров';
  Mas1[1].Age:=55;

  Mas1[2].Name:='Шевченко';
  Mas1[2].Age:=37;
  ..................
```

Для статичного масиву наведена ініціалізація його як типізованої константи. Це зручно використовувати для масивів невеликого розміру і в тому випадку, коли значення елементів не змінюються. Ініціалізація константою динамічного масиву неможлива.

Перед початком роботи із динамічним масивом необхідно ініціалізувати певну кількість комірок масиву (див. приклад, наведений вище) за допомогою процедури `SetLength(M,3)`, яка виділяє пам'ять під три комірки динамічного масиву `M` (рахунок елементів починається з нуля). Потім вони ініціалізуються за допомогою операторів присвоювання (ініціалізувати такий масив як константу неможливо). У наведеному прикладі виділяється три комірки під масив, які заповнюються звичайними операторами присвоєння. Реальні задачі вимагають використання великої кількості даних, які часто модифікуються, тому ініціалізація масивів відбувається за допомогою зчитування даних з файлів.

При роботі з набором даних масиву необхідно виконувати такі операції:

- пошук даних за певним полем;
- додавання даних у масив;
- видалення даних з масиву;
- модифікація даних у масиві.

Частота застосування цих операцій, а також вимоги до тривалості їх виконання у значній мірі залежать від призначення системи, в якій вони використовуються. Розглянемо особливості обробки даних у масивах.

Пошук даних за значенням певного поля. Якщо масив не відсортований (неупорядкований), то єдиний спосіб пошуку даних – це послідовний перегляд всіх елементів, поки не буде знайдено необхідний елемент даних або не досягнуто кінець масиву, що означатиме, що елемент не знайдено. Кількість порівнянь в середньому буде:

`N/2` – коли такий елемент є в масиві;

`N` – коли такий елемент відсутній в масиві;

де `N` – кількість елементів масиву.

Якщо додатково треба визначити і кількість таких елементів, то необхідно переглянути всі `N` елементів.

Розглянемо приклад масиву, кожний елемент якого – дані про людину (запис), який складається з двох полів: прізвище `(Name)` та вік `(Age)`. Масив не відсортований.

Приклад 1.3

Треба з'ясувати, чи є у статичному масиві особа із віком `K` років, визначити тільки наявність такої особи. Для цього можна написати в якості приклада такий програмний код:

```
type
   Person=record //структура елемента масиву
     Name: string[15];
     Age: byte;
   end;
   Mas1=array[1..N] of Person;
var
   M: Mas1;
   I,N,K: byte;
   Flag: boolean;

begin
   {ініціалізація масиву (опущено)}
   ..............................................
   I:=1; N:=3; K:=55;
   Flag:=false;
   while I<=N do
     begin
       if M[I].Age=K then
         begin
           writeln('Особа з віком ', K, ' років у
масиві є, це: прізвище - ', M[I].Name, ' вік - ',
M[I].Age, ' років');
```

```
            Flag:=true;
            break;
          end;
        Inc(I);
      end;
  if not Flag then
    writeln('Такої особи в масиві нема!');
  ..............................................
```

Ситуацію можна суттєво покращати, якщо масив відсортувати за значенням елемента, за яким відбувається пошук. У цьому випадку кількість порівнянь в середньому буде дорівнювати `N/2` як для ситуацій, коли такий елемент є в масиві, так і для ситуацій, коли такий елемент відсутній в масиві.

Для упорядкованого масиве є ще більш ефективний метод пошуку – бінарний. Сутність методу можна представити наступним алгоритмом:

1. Кількість елементів масиву ділиться навпіл і для цього номеру обирається значення ключа елемента.

2. Виконується порівняння значення отриманого ключа з тим, що шукається. Якщо значення співпадають, пошук завершено, а якщо ні, то переходимо до вдвічі меншого масиву, у якому знаходиться елемент, який шукається, після чого переходимо до п. 1.

При цьому кількість порівнянь у найгіршому випадку не перевищить $\log_2 N$. При кількості елементів 1024 кількість порівнянь не перевищить 10, при 1 000 000 – 20, 1 000 000 000 – 30 і т. д. Нижче наведена функція, яка виконує бінарний пошук у відсортованому масиві.

```
function BinarySeekMassiv(Left,Right, Number: integer;
                          var Index,Quant: integer;
                          M: Mas1): boolean;
{Бінарний пошук елемента Number у відсортованому
по збільшенню масиві M, у діапазоні з номером
елемента Left до номера елемента Right, якщо елемент
знайдено BinarySeekMassiv=true інакше false.
Index - номер знайденого елемента.
Quant - кількість порівнянь}
begin
  BinarySeekMassiv:=false; Quant:=0;
  while Left<=Right do
    begin
      Middle:=(Left+Right) div 2; {середина діапазону,
у якому відбувається пошук}
```

```
{якщо значення середнього елемента менше того, який
треба знайти, для нового діапазону пересуваємо ліву
границю на середину попереднього діапазону}
        if M[Middle]<Number then
          begin
            Left:=Middle+1;
            Inc(Quant);
          end
        else {якщо значення середнього елемента менше
того, який треба знайти, то навпаки}
          if M[Middle]>Number then
            begin
              Right:=Middle-1;
              Inc(Quant);
            end
          else //елемент знайдено
            begin
              BinarySeekMassiv:=true;
              Index:=Middle;
              Exit;
            end;
     end; //while Left<=Right do
end; //function BinarySeekMassiv
```

Розглянемо операції модифікація масиву, тобто редагування, видалення та додавання елементів.

Редагування даних у масиві. Це найпростіша операція з усіх, тому що вона зводиться до пошуку відповідного елемента у масиві, як вже описано далі у прикладі 1.3, і модифікації його полів шляхом простого присвоєння нових значень. Ця операція настільки проста, що її код тут не наводиться.

Видалення даних з масиву. Це набагато складніша операція, тому що вона складається із певної низки відносно довгих операцій з масивом, які виконуються за наступним алгоритмом:

Виконати пошук елемента, який потрібно видалити.

Якщо його знайдено, то можна застосувати два шляхи:

1. Починаючи із номера знайденого елемента в циклі виконується пересув елементів на один ліворуч (в напрямку до початку масиву) до кінця масиву, таким чином з послідовності зникає елемент, який необхідно видалити. При цьому не залишається порожнього елемента в масиві, але треба пам'ятати, що процес обміну елементів при зсуві вимагає набагато більше часу, ніж порівняння.

2. У структурі елемента заздалегідь необхідно передбачити поле, яке буде означати: дійсний це елемент або виключений із послідовності. У знайденому елементі робиться відмітка про його виключення з послідовності. Перевага цього підходу полягає в тому, що можна уникнути такої довготривалої операції, як переміщення елементів. Цей алгоритм можна застосовувати як для несортованих масивів, так і для сортованих. Недоліком є те, що залишаються «дірки» у масиві, якщо їх буде багато, то обробка масиву суттєво уповільниться. При будь-якій операції з даними необхідно кожний елемент перевіряти на дійсність.

Додавання даних у масив. Якщо масив невідсортований і динамічний, то додавання нового елемента зводиться до збільшення кількості елементів на одиницю (1) і додаванню його у кінець. Для статичного масиву додавання нового елемента можливо тільки у випадку, якщо є вільні порожні елементи масиву в кінці, або «дірки» при попередньому видаленні елемента шляхом позначення його як недійсного.

Якщо масив відсортований, то можна застосувати два підходи:

1. Знайти місце вставки, зсунути елементи праворуч на 1, записати новий елемент на звільнене місце.

2. Дописати новий елемент в окремий додатковий масив. При необхідності пошуку спочатку виконати пошук в основному масиві бінарним алгоритмом, а потім у додатковому – звичайним послідовним. Якщо об'єм основного масиву достатньо великий, а додаткового – навпаки, то цей підхід може суттєво зменшити час роботи програми. Через певний відрізок часу, звичайно, ці два масиви об'єднують в один масив та знову відсортовують його (ця операція має назву – реорганізація).

Розглянемо простий приклад додавання ще одного непарного елемента у відсортований масив парних цілих чисел.

Приклад 1.4

Статичний масив `M` має `N=7` елементів, перші шість (6) з яких ініціалізовано наступним чином: {2,4,6,8,10,12}. Треба додати до масиву число `K=9`, не порушуючи упорядкованості. Код для цієї операції може бути наступний:

```
I:=1;
while M[I]<K do Inc(I); {I - номер елемента, на
  місце якого треба вставити число K}
for J:=N downto I do {зсув елементів масиву на 1
  праворуч}
  M[J]:=M[J-1];
  M[I]:=K; //вставка нового елемента
```