1.6. ДИНАМІЧНІ СПИСКИ

Під списком розуміють послідовність елементів, у якій наступний елемент пов'язаний з попереднім за допомогою вказівника. Це означає, що елемент структури окрім інформаційної частини повинен обов'язково мати окреме поле, у якому буде зберігатися вказівник на наступний елемент. Звідси зрозуміло, що єдиною структурою елемента списку може

бути запис (record), який дозволяє описувати у себе статичні поля різних типів.

Динамічні списки формуються за допомогою вказівника – структури алгоритмічної мови, яка містить адресу комірки в оперативній пам'яті комп'ютера. У зв'язку з тим, що час переходу до адреси за вказівником не залежить від місця розташування комірки у пам'яті, то фізично елементи списку можуть бути розташовані хаотично у доступному просторі. Але основна перевага цієї структури порівняно з масивами полягає у тому, що операції додавання елемента, або його видалення не пов'язані із необхідністю зсуву елементів, при цьому не залишається «дірок» у структурі даних. За такою можливістю для відсортованих послідовностей можна легко виконувати як додавання нового елемента у послідовність, так і вилучення вже непотрібного елемента з неї, не порушуючи упорядкованості. Після видалення елемента з послідовності пам'ять, яку він займав, звільнюється, і вона може бути використана операційною системою знову. Схему операції додавання нового елемента до відсортованого списку зображено на рисунку 1.7 (новий елемент виділено жирною рамкою).

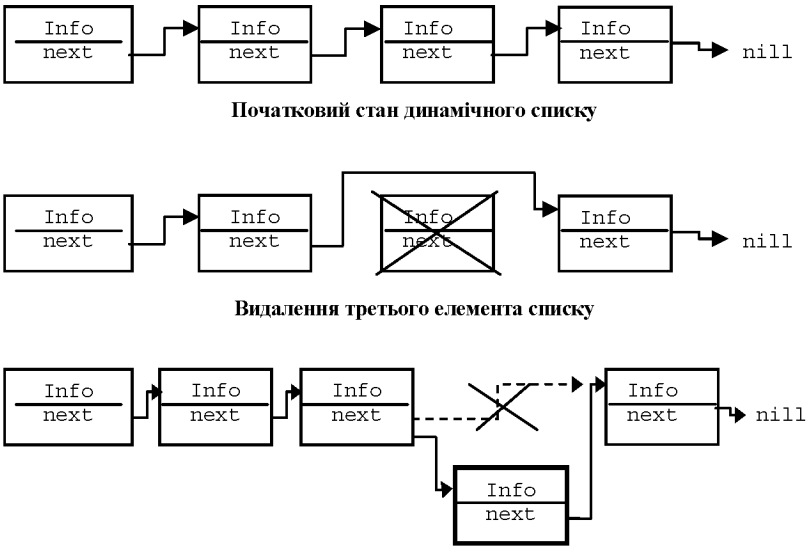

Рис. 1.7. Схематичне представлення операцій зі списком

Операція виконується у дві дії:

1. Пошук елемента, перед яким або після якого треба робити вставку.

2. Зміна адреси (переадресація) елементів списку таким чином, щоб зберегти зв'язність списку.

При такому підході не порушується упорядкованість послідовності. Аналогічним чином виконується видалення елемента списку, не порушуючи його зв'язності.

Модифікація даних відбувається звичайним шляхом, переадресація при цьому непотрібна. Нижче наведено код для формування динамічного списку та його модифікації. Дії, які при цьому виконуються, зрозумілі з коментарів до операторів фрагменту програми.

```
type
   One=^Person;
   Person=record //опис структури елемента списку
      Name: string[20];
      Year: byte;
      Next: One; //вказівник на наступний елемент
   end;
   Group=array[1..5] of Person; //масив елементів списку
var
   P,Po,Pn,PP,First: One;
   I: byte;
   Key: string; {ключ, за яким буде відбуватися пошук
        елемента}
const {ініціалізація масиву записів, які будуть
   складати список}
   G: Group=((Name:'Smith'; Year:35),
             (Name:'Ivanoff'; Year:47),
             (Name:'Nemirov'; Year:48),
             (Name:'Petrov'; Year:65),
             (Name:'Shevchenko'; Year:25));
begin
   New(P); {виділення пам'яті під елемент списку}
   First:=P;  {запам'ятати адресу першого елемента
списку}
   for I:=1 to 5 do
      begin {заповнення списку з масиву записів}
         P^.Name:=G[I].Name;
         P^.Year:=G[I].Year;
```

```
        if I<5 then
           begin
              New(P^.Next); {зв'язок поточного елемента
                 із наступним}
              P:=P^.Next;
           end; //if I <5 then
     end; //for I:=1 to 5 do
     P^.Next:=nil; //формування ознаки кінця списку
     //-------------------------------------------------
     P:=First; //виведення елементів списку на екран
     while P<>nil do
        begin
           writeln(P^.Name, '  ', P^.Year);
           P:=P^.Next; {просування по списку – тільки
              вперед}
        end; //while P<>nil do
     //-------------------------------------------------
     writeln('*******insert element after key=
''Nemirov''****');
        {вставка елемента списку після елемента із ключем
'cccccc'}
     P:=First; {просування по списку і зупинка на
        елементі із ключем Nemirov}
     while (P^.Name<>'Nemirov') and (P<>nil) do
        P:=P^.Next;
     {формування і вставка нового елемента у список}
     if P<>nil then
        begin
           New(Pn); {формування нового елемента і
заповнення полів}
           Pn^.Name:='Nord';
           Pn^.Year:=105;
           Pn^.Next:=P^.Next; {переадресація, див.
              рисунок, попередній зв'язок пропадає}
           P^.Next:=Pn;
        end;
     //-------------------------------------------------
     P:=First;
     while P<>nil do
        begin {пошук елемента, який необхідно видалити
           та зупинка перед ним}
```

```
            writeln(P^.Name, '   ', P^.Year);
            P:=P^.Next;
         end;
      writeln('*****deleting element with key=Nemirov
*****');
      P:=First; Key:='Nemirov';
      while (P^.Next<>nil) and (P^.Next^.Name <>Key) do
         P:=P^.Next;
         {видалення елемента Key і звільнення пам'яті
під нього}
         if P^.Next<>nil then
            begin
               Po:=P^.Next; //переадресація
               P^.Next:=P^.Next^.Next;
               Dispose(Po); //звільнення пам'яті
               P:=First;
               while P<>nil do
                  begin //виведення списку на екран
                     writeln(P^.Name, '   ', P^.Year);
                     P:=P^.Next;
                  end;
            end
         else writeln('There is no element ', Key,
' in list');
```

Така структура даних дозволяє найбільш раціонально організувати процес розподілення оперативної пам'яті при роботі з даними, тому що дозволяє виділяти та звільняти пам'ять при модифікації структури даних (додавання та видалення елементів).

Елементи послідовності можна легко сортувати за певним ключем, але, на жаль, навіть до відсортованого списку не можна застосувати бінарний метод пошуку. Причина у тому, що процес просування по елементам послідовності – тільки послідовний, починаючи з першого елемента. На відміну від масивів та файлів тут немає можливості отримати доступ до елемента з певним номером (прямий доступ). Але треба відмітити, що процес просування по елементах достатньо швидкий.

Нижче наведено код функції сортування списку по одному полю.

```
procedure SortOne();
{Сортування лінійного списку по одному полю (по
пріоритету) простим обміном ("бульбашкою"). Елемент
```

```
списку складається з двох полів: p - пріоритет заявки,
q - обсяг заявки}
var
  NewP,NewFirst: TNode;
  P: Node; //допоміжна змінна для обміну
begin
  NewFirst:=First; //перший елемент списку
  while NewFirst^.Next<>nil do
    begin {зовнішній цикл перебору всіх елементів
списку}
      NewP:=NewFirst^.Next;
      while NewP<>nil do
        begin {внутрішній цикл перебору всіх елементів
списку}
          if NewFirst^.p>NewP^.p then
            begin {якщо наступний елемент менше
попереднього - обмін через допоміжну змінну}
              P.p:=NewP^.p;
              P.q:=NewP^.q;
              NewP^.p:=NewFirst^.p;
              NewP^.q:=NewFirst^.q;
              NewFirst^.p:=P.p;
              NewFirst^.q:=P.q;
            end;
          NewP:=NewP^.Next;
        end;
      NewFirst:=NewFirst^.Next;
    end;
end;
```