Лекція 10. Динамічні структури даних. Лінійні списки

Лінійні списки

Для роботи з динамічними структурами даних використовуються вказівники. Вказівники являють собою спеціальний тип даних. Вони приймають значення, рівні адресам розміщення в оперативній пам'яті відповідних динамічних змінних.

Списком називається структура даних, кожний елемент якої за допомогою вказівника зв'язується з наступним елементом. На найперший елемент (голову списку) є окремий вказівник.

З визначення треба, що кожний елемент списку містить поле даних (воно може мати складну структуру) і поле посилання на наступний елемент. Після посилання останнього елемента повинне містити порожній вказівник (nil).

Найбільш простою динамічною структурою є односпрямований список, елементами якого служать об'єкти *структурного типу*.

Односпрямований (однозв'язний) список – це структура даних, що представляє собою послідовність елементів, у кожному з яких зберігається значення й вказівник на наступний елемент списку (рис. 1). В останньому елементі вказівник на наступний елемент дорівнює NULL.

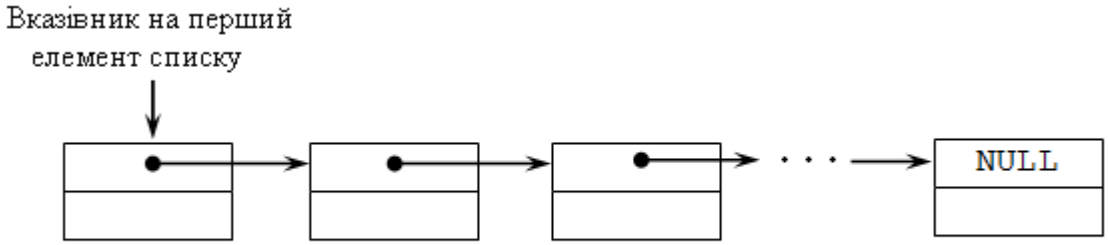


Рис. 1. Лінійний односпрямований список

Опис найпростішого елемента такого списку виглядає в такий спосіб:
struct ім'я_типу { інформаційне поле; адресне поле; };
де інформаційне поле – це поле кожного, раніше оголошеного або стандартного, типу;
адресне поле – це вказівник на об'єкт того ж типу, що й обумовлена структура, у нього записується адреса наступного елемента списку.
Наприклад:

```
struct Node {
             int key;//інформаційне поле
             Node*next;//адресне поле
            };
Інформаційних полів може бути кілька.
Наприклад:
struct point {
              char*name;//інформаційне поле
              int age;//інформаційне поле
              point*next;//адресне поле
             };
```

Кожний елемент списку містить ключ, що ідентифікує цей елемент. Ключ звичайно буває або цілим числом, або рядком.
Основними операціями, здійснюваними з односпрямованими списками, є:

- створення списку;
- печатка (перегляд) списку;
- вставка елемента в список;
- видалення елемента зі списку;
- пошук елемента в списку
- перевірка порожнечі списку;
- видалення списку.

Особлива увага варто звернути на те, що при виконанні будь-яких операцій з лінійним односпрямованим списком необхідно забезпечувати позиціонування якого-небудь вказівника на перший елемент. У противному випадку частина або весь список буде недоступний.
Розглянемо докладніше кожну з наведених операцій.

Число елементів зв'язаного списку може рости або зменшуватися залежно від того, скільки даних ми хочемо зберігати в ньому. Щоб додати новий елемент у список, необхідно:

1. Одержати пам'ять для нього;
2. Помістити туди інформацію;
3. Додати елемент у кінець списку (або початок).

Елемент списку складається з різнотипних частин (збережена інформація й вказівник), і його природно представити записом. Перед описом самого запису описують вказівник на неї:

```
Type { опис списку із цілих чисел }
  PList = ^TList;
  TList = record
     Inf : Integer;
     Next : PList;
  end;
```

Приклади

Створення списку.

Завдання. Сформувати список, що містить цілі числа 3, 5, 1, 9.

Визначимо запис типу TList з полями, що містять характеристики даних - значення чергового елемента й адреси наступного за ним елемента

```
PList = ^TList;
  TList = record
     Data : Integer;
     Next : PList;
  end;
```

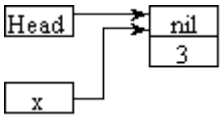
Щоб список існував, треба визначити вказівник на його початок. Опишемо змінні.

```
Var
  Head, x : PList;
```

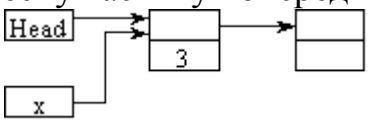
Створимо перший елемент: New(Head); { виділяємо місце в пам'яті для змінної Head } Head^.Next := nil; { вказівник на наступний елемент порожній (такого елемента немає) } Head^.Data := 3; { заповнюємо інформаційне поле першого елемента }

Продовжимо формування списку, для цього потрібно додати елемент у кінець списку.

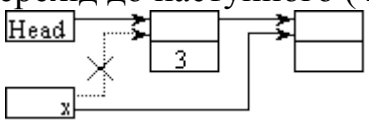
Уведемо допоміжну змінну вказівного типу, що буде зберігати адреса останнього елемента списку: x := Head; {зараз останній елемент списку збігається з його початком}



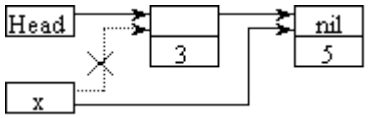
New(x^.Next); { виділимо області пам'яті для наступні (2-го) елементи й помістимо його адресу в адресну частину попередні (1-го) елемента }



x := x^.Next ; { змінна x приймає значення адреси виділеної області. У такий спосіб здійснюється перехід до наступного (2-ому) елементу списку }



x^.Data := 5; { значення цього елемента } x^.Next := nil; {наступного значення немає }



Інші числа заносяться аналогічно: New(x^.Next); { виділимо області пам'яті для наступного елемента } x := x^.Next ; { перехід до наступного (3-му) елементу списку } x^.Data := 1; { значення цього елемента } x^.Next := nil; {наступного значення немає } New(x^.Next); { виділимо області пам'яті для наступного елемента } x := x^.Next ; { перехід до наступного (4-му) елементу списку } x^.Data := 9; { значення цього елемента } x^.Next := nil; {наступного значення немає }

Зауваження. Як видно із приклада, відмінним є тільки створення першого (Head) елемента – голови списку. Всі інші дії повністю аналогічні і їх природно виконувати в циклі.

Приєднання нового елемента до голови списку провадиться аналогічно: ...……..... New(x); { уведення значення елемента x^.Data := ... } x^.Next := Head; Head := x; ...…….....

У цьому випадку останній уведений елемент виявиться в списку першим, а перший - останнім.

Перегляд списку.

Перегляд елементів списку здійснюється послідовно, починаючи з його початку. Вказівник List послідовно посилається на перший, другий і т.д.

елементи списку доти, поки весь список не буде пройдений. При цьому з кожним елементом списку виконується деяка операція- наприклад, печатка елемента. Початкове значення List - адреса першого елемента списку (Head). Digit - значення елемента, що видаляється.

```
List := Head;
While List^.Next <> nil do
  begin
    WriteLn(List^.Data);
    List := List^.Next; { перехід до наступного елемента; аналог для масиву i:=i+1 }
  end;
```

Видалення елемента зі списку.

При видаленні елемента зі списку необхідно розрізняти три випадки:1. Видалення елемента з початку списку.

2. Видалення елемента із середини списку.

3. Видалення з кінця списку.

Видалення елемента з початку списку.

```
List := Head; { запам'ятаємо адресу першого елемента списку }
Head := Head^.List; { тепер Head указує на другий елемент списку }
Dispose(List); { звільнимо пам'ять, зайняту змінної List^ }
```

Видалення елемента із середини списку.

Для цього потрібно знать адреси елемента, що видаляється, і елемента, що перебуває в списку перед ним.

```
List := Head;
While (List<>nil) and (List^.Data<>Digit) do
  begin
    x := List;
    List := List^.Next;
  end;
x^.Next := List^.Next;
Dispose(List);
```

Видалення з кінця списку.

Воно провадиться, коли вказівник x показує на передостанній елемент списку, а List - на останній.

```
List := Head; x := Head;
While List^.Next<>nil do
  begin
    x := List;
    List := List^.Next;
  end;
x^.Next := nil;
Dispose(List);
```

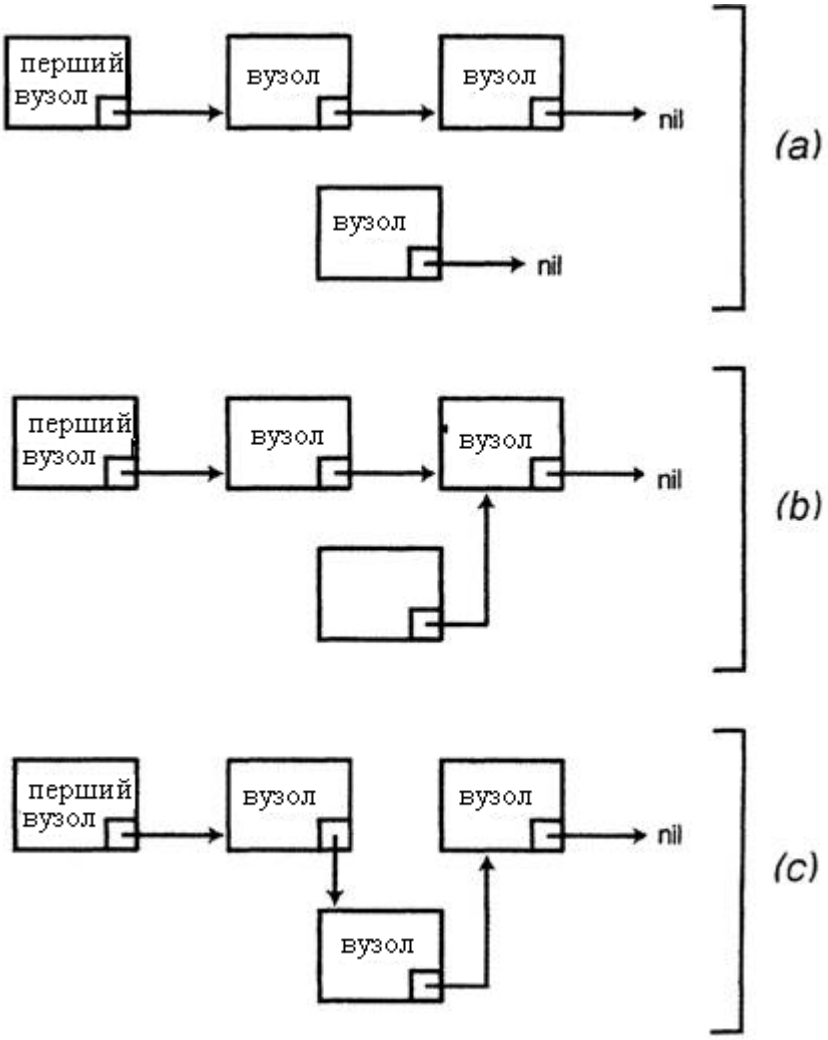


Рис. 2. Вставка нового вузла в однозв'язний список

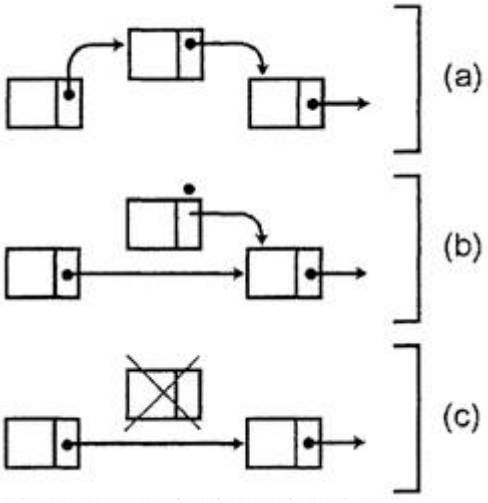


Рис. 3. Видалення вузла з однозв'язного списку