



Динамические списки

Алтайский государственный университет
Математический факультет
Кафедра информатики
Барнаул 2014

Лекция 20

■ План

□ Динамические списки

- Динамические структуры данных
- Списки: состав функций
- Создание, добавление, поиск, удаление узлов
- Решение задач с использованием списков



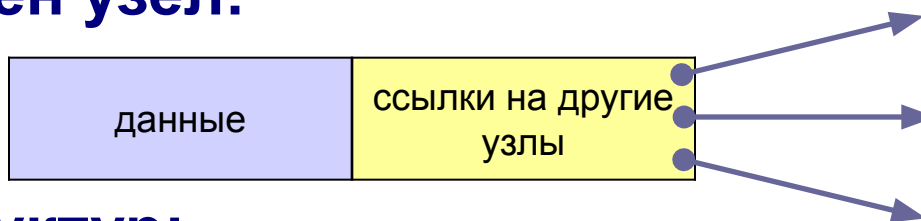
Динамические списки

- Динамические структуры данных
- Списки: состав функций
- Создание, добавление, поиск, удаление узлов
- Решение задач с использованием списков

Динамические структуры данных

Строение: набор узлов, объединенных с помощью **ССЫЛОК**.

Как устроен узел:



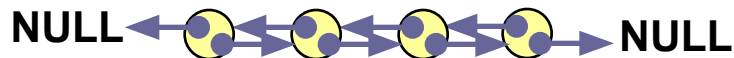
Типы структур:

СПИСКИ

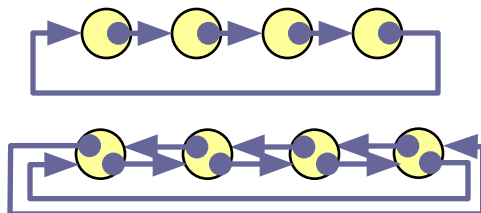
односвязный



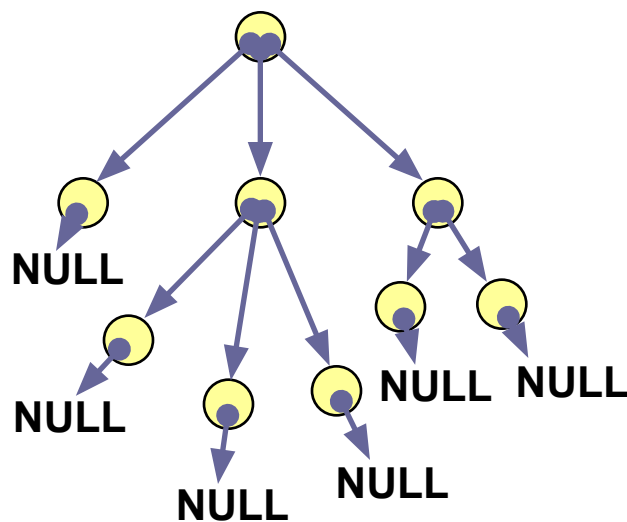
двунаправленный (двусвязный)



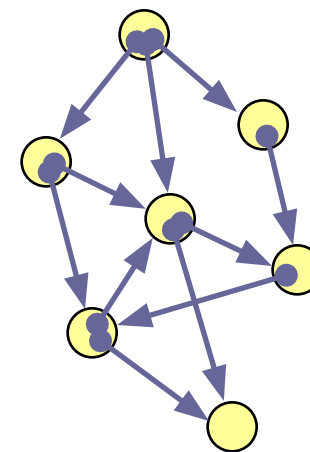
циклические списки (кольца)



деревья



графы



Когда нужны списки?

Задача (алфавитно-частотный словарь). В файле записан текст. Нужно записать в другой файл в столбик все слова, встречающиеся в тексте, в алфавитном порядке, и количество повторений для каждого слова.

Проблемы:

- 1) количество слов заранее неизвестно (~~статический массив~~);
- 2) количество слов определяется только в конце работы (~~динамический массив~~).

Решение – список.

Алгоритм:

- 3) создать список;
- 4) если слова в файле закончились, то стоп.
- 5) прочитать слово и искать его в списке;
- 6) если слово найдено – увеличить счетчик повторений, иначе добавить слово в список;
- 7) перейти к шагу 2.

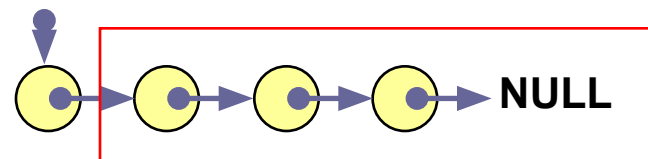
СПИСКИ: НОВЫЕ ТИПЫ ДАННЫХ

Что такое список:

- 1) пустая структура – это список;
- 2) список – это начальный узел (*голова*) и связанный с ним список.



Рекурсивное определение!



Структура узла:

```
struct Node {  
    char word[40];    // слово  
    int  count;      // счетчик повторений  
    Node *next;      // ссылка на следующий элемент  
};
```

Указатель на эту структуру:

```
typedef Node *PNode;
```

Адрес начала списка:

```
PNode Head = NULL;
```



Для доступа к списку достаточно знать адрес его головы!

Что нужно уметь делать со списком?

1. **Создать** новый узел.
2. **Добавить** узел:
 - a) в начало списка;
 - b) в конец списка;
 - c) после заданного узла;
 - d) до заданного узла.
3. **Искать** нужный узел в списке.
4. **Удалить** узел.

Создание узла

Функция `CreateNode` (*создать узел*):

ВХОД: новое слово, прочитанное из файла;

ВЫХОД: адрес нового узла, созданного в памяти.

возвращает адрес
созданного узла

НОВОЕ СЛОВО

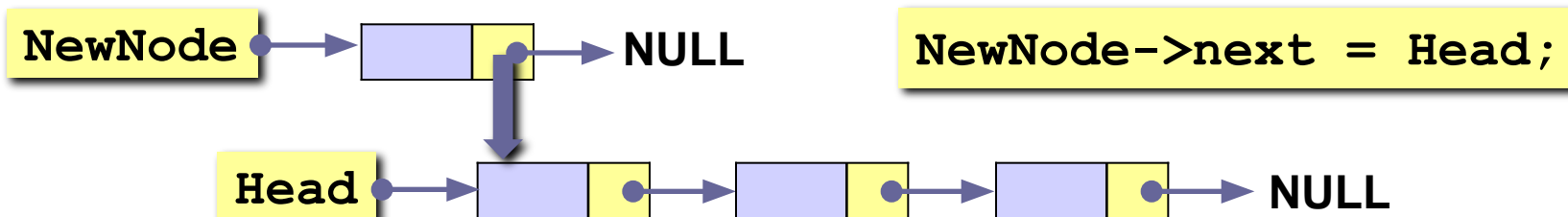
```
PNode CreateNode ( char NewWord[] )
{
    PNode NewNode = new Node;
    strcpy (NewNode->word, NewWord);
    NewNode->count = 1;
    NewNode->next = NULL;
    return NewNode;
}
```



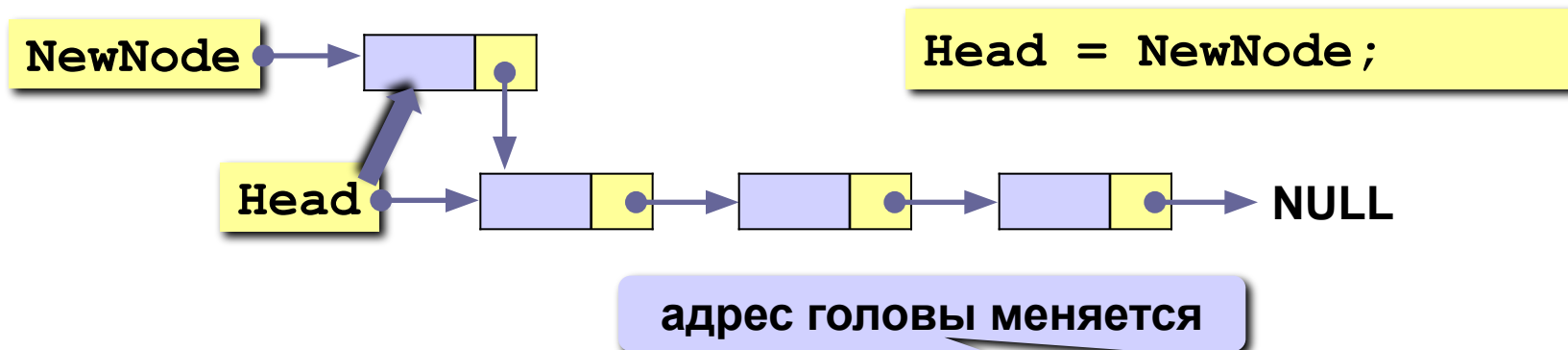
Если память
выделить не
удалось?

Добавление узла в начало списка

1) Установить ссылку нового узла на голову списка:



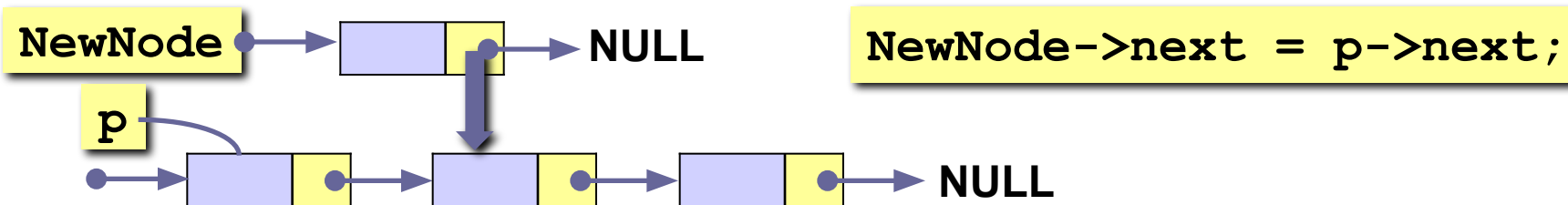
2) Установить новый узел как голову списка:



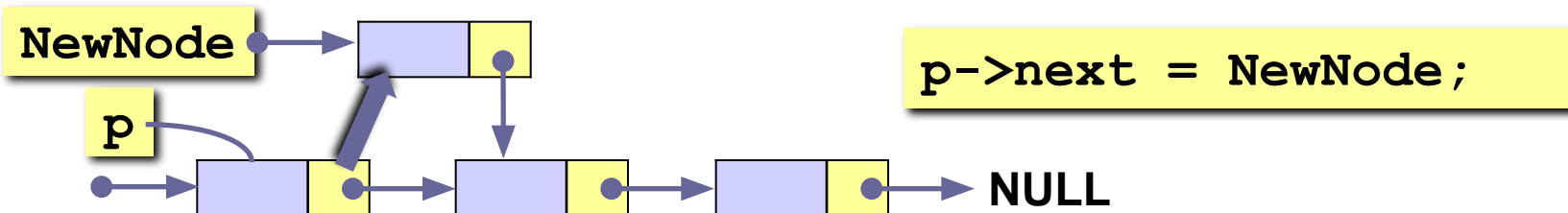
```
void AddFirst (PNode * Head, PNode NewNode)
{
    NewNode->next = *Head;
    *Head = NewNode;
}
```

Добавление узла после заданного

1) Установить ссылку нового узла на узел, следующий за p:



2) Установить ссылку узла p на новый узел:

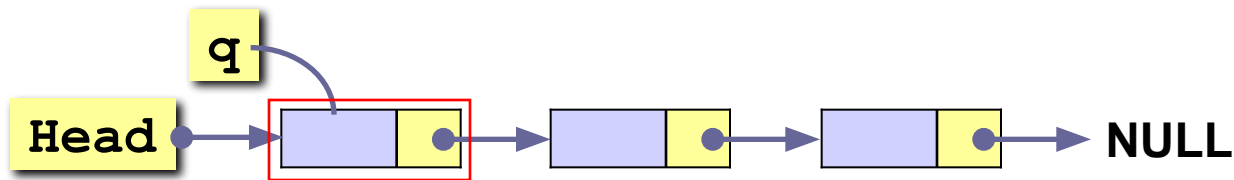


```
void AddAfter (PNode p, PNode NewNode)
{
    NewNode->next = p->next;
    p->next = NewNode;
}
```

Проход по списку

Задача:

сделать что-нибудь хорошее с каждым элементом списка.



Алгоритм:

- 1) установить вспомогательный указатель **q** на голову списка;
- 2) если указатель **q** равен **NULL** (дошли до конца списка), то стоп;
- 3) выполнить действие над узлом с адресом **q** ;
- 4) перейти к следующему узлу, **q->next**.

```
...  
PNode q = Head;           // начали с головы  
while ( q != NULL ) {    // пока не дошли до конца  
    ...                   // делаем что-то хорошее с q  
    q = q->next;          // переходим к следующему узлу  
}  
...
```

Добавление узла в конец списка

Задача: добавить новый узел в конец списка.

Алгоритм:

- 1) найти последний узел **q**, такой что **q->next** равен **NULL**;
- 2) добавить узел после узла с адресом **q** (процедура **AddAfter**).

Особый случай: добавление в пустой список.

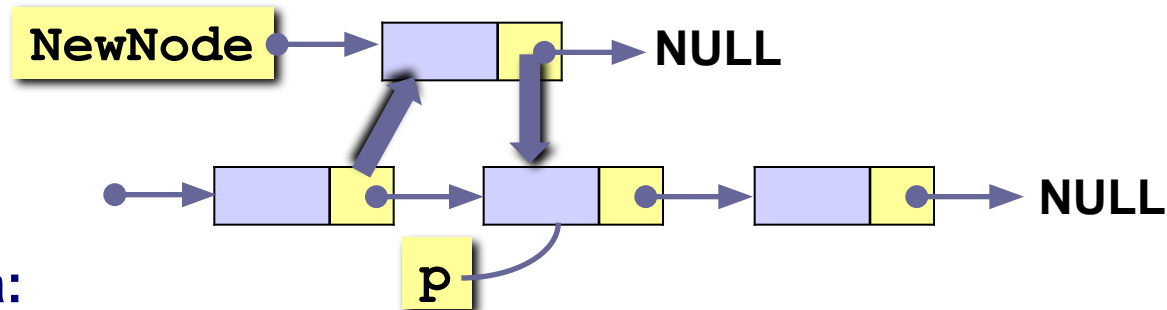
```
void AddLast ( PNode *Head, PNode NewNode )
{
  PNode q = *Head;
  if ( *Head == NULL ) {
    AddFirst( Head, NewNode );
    return;
  }
  while ( q->next ) q = q->next;
  AddAfter ( q, NewNode );
}
```

особый случай – добавление в пустой список

ищем последний узел

добавить узел
после узла q

Добавление узла перед заданным



Проблема:

нужно знать адрес **предыдущего** узла, а идти назад нельзя!

Решение: найти предыдущий узел **q** (проход с начала списка).

```
void AddBefore ( PNode * Head, PNode p, PNode NewNode )
{
    PNode q = *Head;
    if ( *Head == p ) {
        AddFirst ( Head, NewNode );
        return;
    }
    while ( q && q->next != p ) q = q->next;
    if ( q ) AddAfter(q, NewNode);
}
```

особый случай – добавление в начало списка

ищем узел, следующий за которым – узел p

добавить узел после узла q

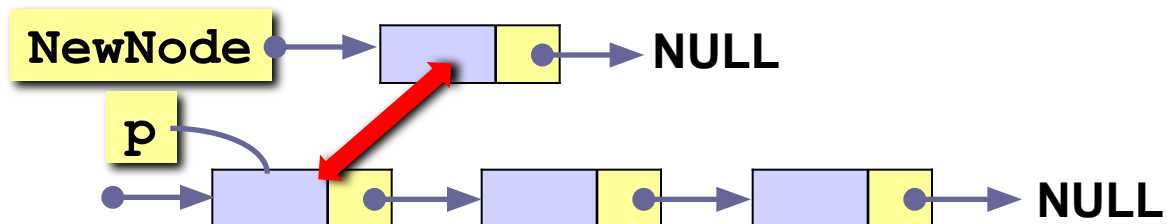
? Что плохо?

Добавление узла перед заданным (II)

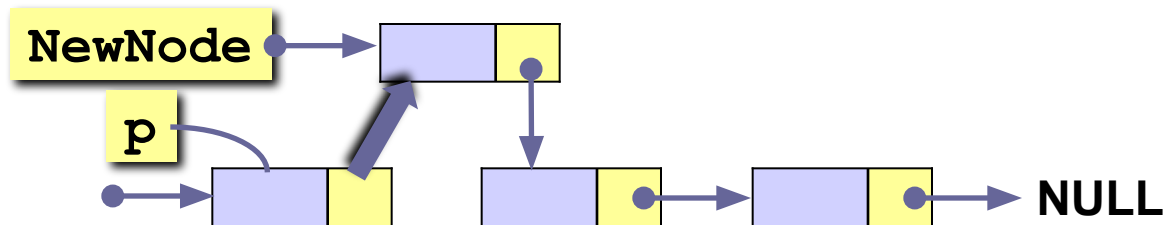
Задача: вставить узел перед заданным без поиска предыдущего.

Алгоритм:

- 1) поменять местами данные нового узла и узла **p**;



- 2) установить ссылку узла **p** на **NewNode**.



```
void AddBefore2 ( PNode p, PNode NewNode )
{
    Node temp;
    temp = *p; *p = *NewNode;
    *NewNode = temp;
    p->next = NewNode;
}
```



Так нельзя, если $p == \text{NULL}$ или адреса узлов где-то еще запоминаются!

Поиск слова в списке

Задача:

найти в списке заданное слово или определить, что его нет.

Функция Find:

ВХОД: слово (символьная строка);

ВЫХОД: адрес узла, содержащего это слово или **NULL**.

Алгоритм: проход по списку.

результат – адрес узла

ИЩЕМ ЭТО СЛОВО

```
PNode Find ( PNode Head, char NewWord[] )
{
    PNode q = Head;
    while ( q && strcmp ( q->word, NewWord) )
        q = q->next;
    return q;
}
```

пока не дошли до
конца списка и слово
не равно заданному

Куда вставить новое слово?

Задача:

найти узел, перед которым нужно вставить, заданное слово, так чтобы в списке сохранился алфавитный порядок слов.

Функция `FindPlace`:

ВХОД: слово (символьная строка);

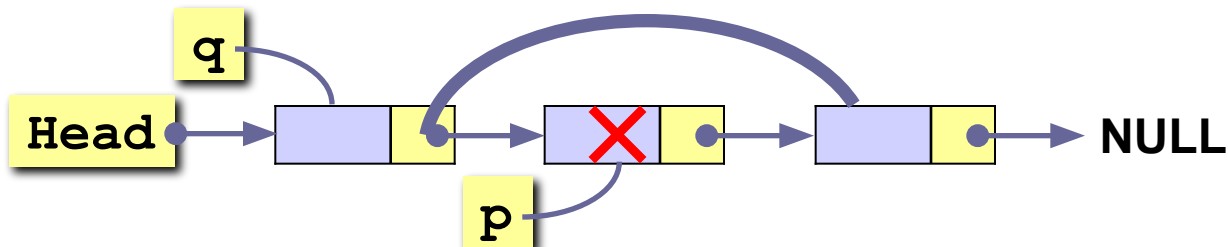
ВЫХОД: адрес узла, перед которым нужно вставить это слово или **NULL**, если слово нужно вставить в конец списка.

```
PNode FindPlace ( PNode Head, char NewWord[] )
{
    PNode q = Head;
    while ( q && strcmp(NewWord, q->word) > 0 )
        q = q->next;
    return q;
}
```

слово `NewWord` стоит по алфавиту до `q->word`

Удаление узла

Проблема: нужно знать адрес предыдущего узла q .



```
void DeleteNode ( Pnode *Head, PNode p )
{
  PNode q = *Head;
  if ( *Head == p )
    *Head = p->next;
  else {
    while ( q && q->next != p )
      q = q->next;
    if ( q == NULL ) return;
    q->next = p->next;
  }
  delete p;
}
```

особый случай:
удаляем первый
узел

ищем предыдущий
узел, такой что
 $q->next == p$

освобождение памяти

Алфавитно-частотный словарь

Алгоритм:

- 1) открыть файл на чтение;

read,
чтение

```
FILE *in;  
in = fopen ( "input.dat", "r" );
```

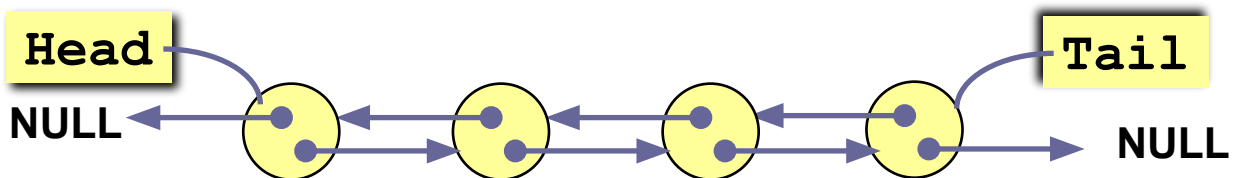
- 2) прочесть слово:

ВВОДИТСЯ ТОЛЬКО ОДНО
слово (до пробела)!

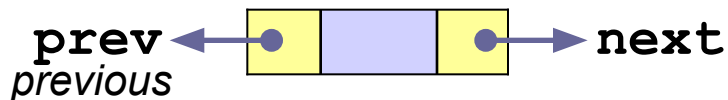
```
char word[80];  
...  
n = fscanf ( in, "%s", word );
```

- 3) если файл закончился ($n \neq 1$), то перейти к шагу 7;
- 4) если слово найдено, увеличить счетчик (поле **count**);
- 5) если слова нет в списке, то
 - создать новый узел, заполнить поля (**CreateNode**);
 - найти узел, перед которым нужно вставить слово (**FindPlace**);
 - добавить узел (**AddBefore**);
- 6) перейти к шагу 2;
- 7) вывести список слов, используя проход по списку.

Двусвязные списки



Структура узла:



```
struct Node {
    char word[40]; // слово
    int count; // счетчик повторений
    Node *next; // ссылка на следующий элемент
    Node *prev; // ссылка на предыдущий элемент
};
```

Указатель на эту структуру:

```
typedef Node *PNode;
```

Адреса «головой» и «хвоста»:

```
PNode Head = NULL;
PNode Tail = NULL;
```



МОЖНО ДВИГАТЬСЯ В обе стороны



нужно правильно работать с двумя указателями вместо одного

Вопросы?

- **Динамические списки**
 - Динамические структуры данных
 - Списки: состав функций
 - Создание, добавление, поиск, удаление узлов
 - Решение задач с использованием списков

