

Динамические структуры данных

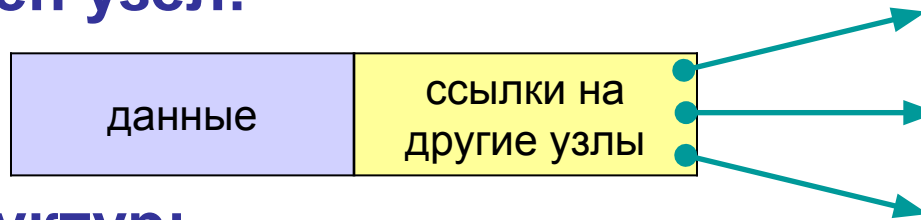
Лекция 5

- Динамические структуры
- Односвязные (однонаправленные) списки
- Двусвязные (двунаправленные) списки

Динамические структуры данных

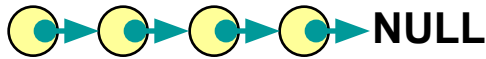
Строение: набор узлов, объединенных с помощью **ССЫЛОК**.

Как устроен узел:



Типы структур:

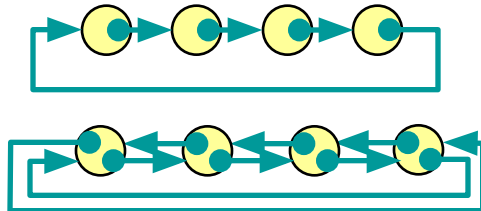
СПИСКИ
односвязный



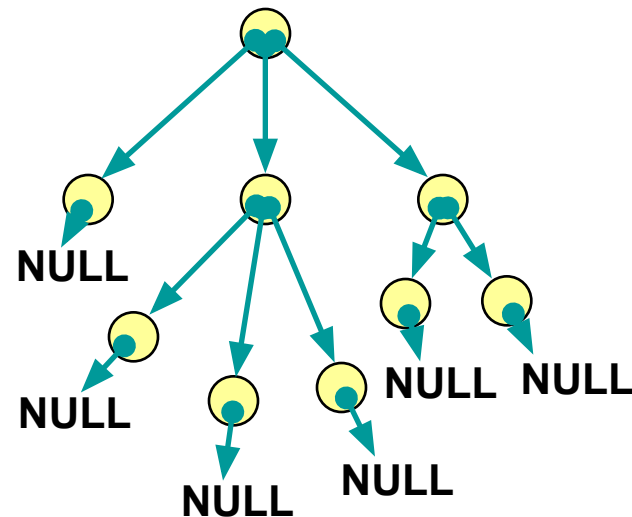
двунаправленный (двусвязный)



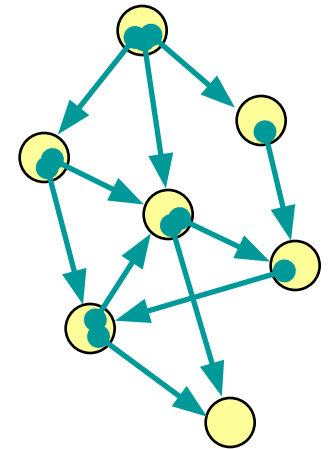
циклические списки (кольца)



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.

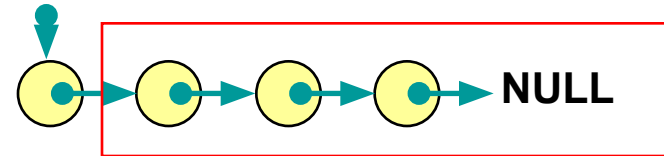
Списки: новые типы данных

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

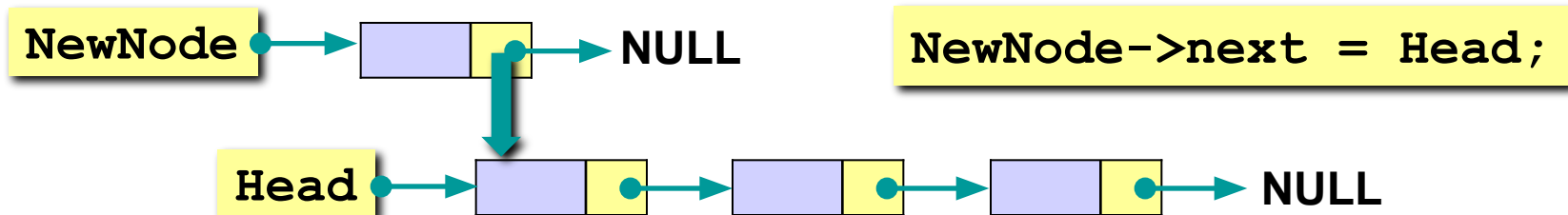
```
PNode CreateNode ( char NewWord[] )
{
    PNode NewNode = new Node;
    strcpy (NewNode->word, NewWord);
    NewNode->count = 1;
    NewNode->next = NULL;
    return NewNode;
}
```



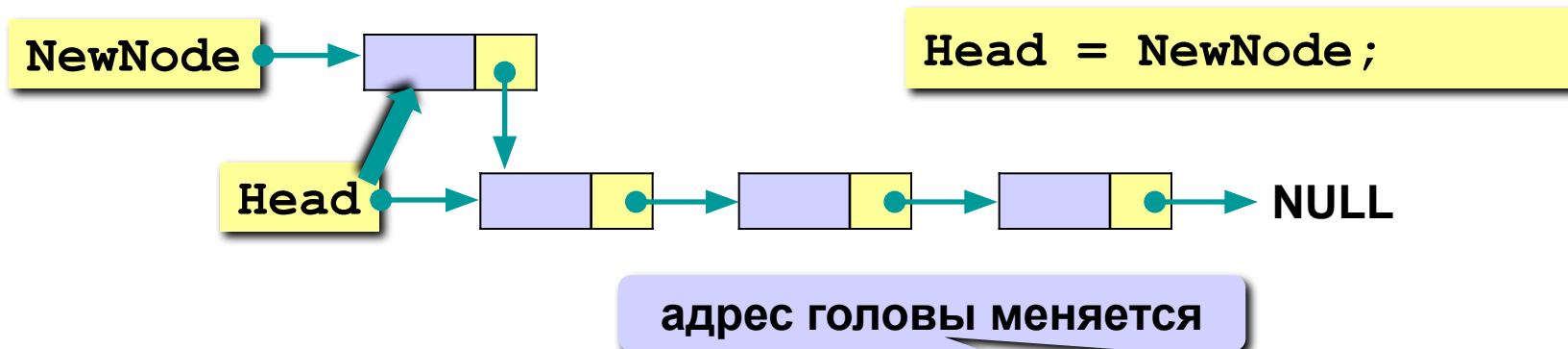
Если память
выделить не
удалось?

Добавление узла в начало списка

1) Установить ссылку нового узла на голову списка:



2) Установить новый узел как голову списка:



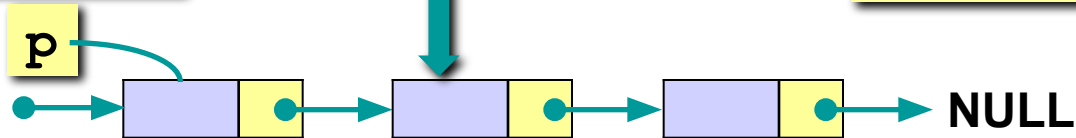
```
void AddFirst (PNode & Head, PNode NewNode)
{
    NewNode->next = Head;
    Head = NewNode;
}
```


Добавление узла после заданного

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

NewNode → [] → NULL

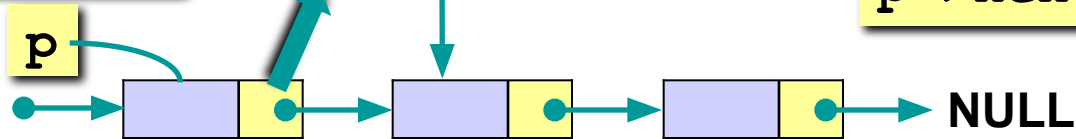
`NewNode->next = p->next;`



2) Установить ссылку узла p на новый узел:

NewNode → []

`p->next = NewNode;`

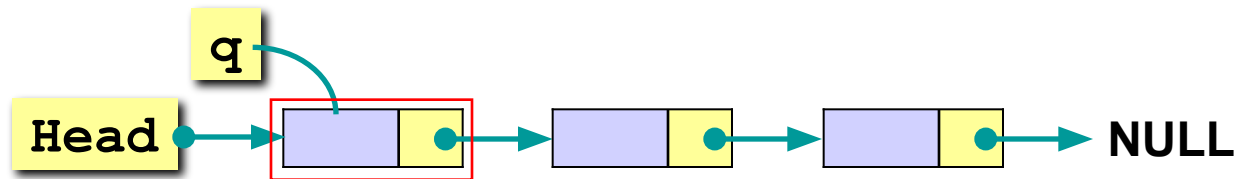


```
void AddAfter (PNode p, PNode NewNode)
{
    NewNode->next = p->next;
    p->next = NewNode;
}
```

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 )
```

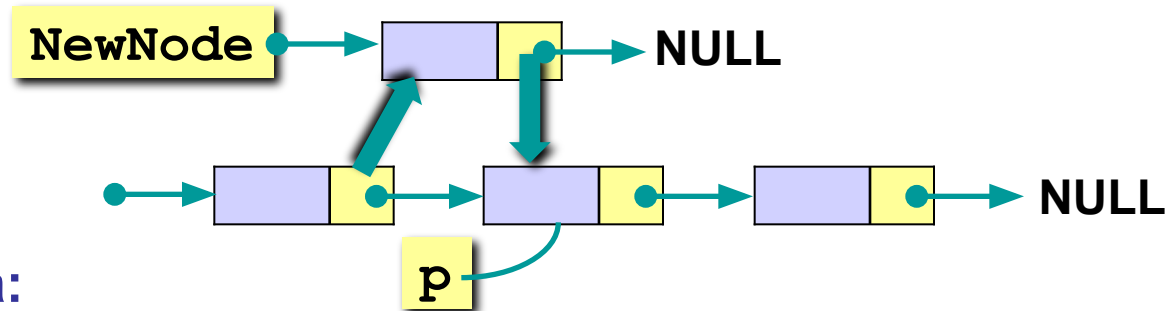
```
{  
  PNode q = Head;  
  if ( Head == NULL )  
    AddFirst ( Head, NewNode );  
  else  
  {  
    while ( q->next ) q = q->next;  
    AddAfter ( q, NewNode );  
  }  
}
```

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 );
  else
  {
    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



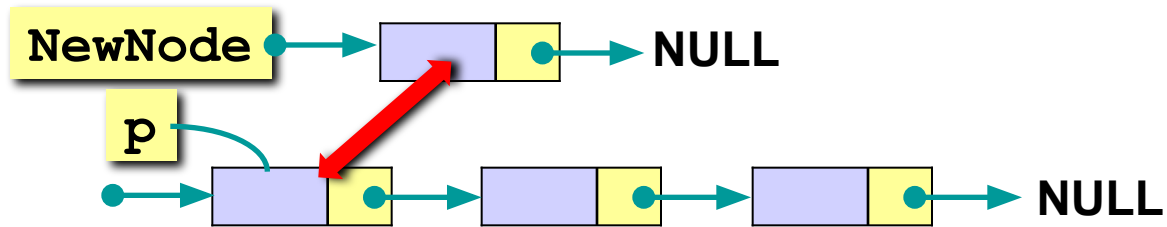
Что плохо?

Добавление узла перед заданным (II)

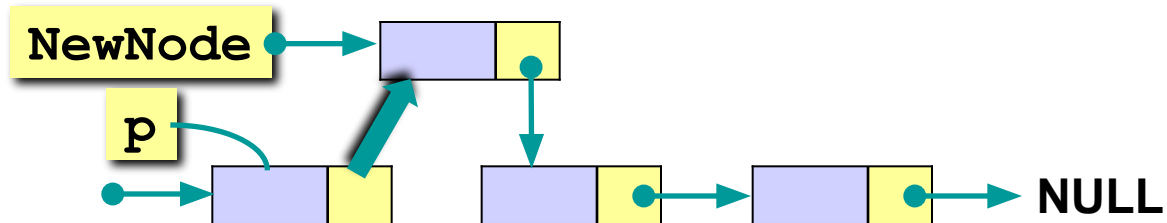
Задача: вставить узел перед заданным без поиска предыдущего.

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);

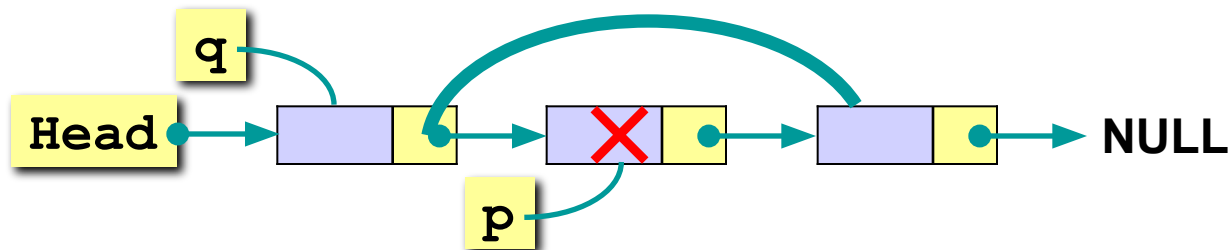
ВЫХОД: адрес узла, перед которым нужно вставить это слово или **NULL**, если слово нужно вставить в конец списка.

```
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;
```

```
}
```

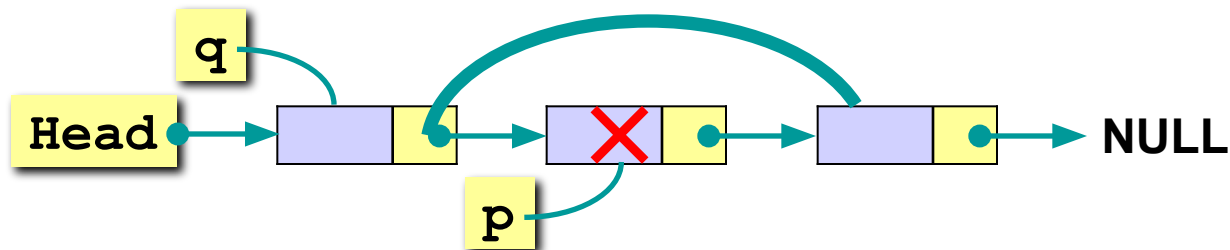
особый случай:
удаляем первый
узел

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

освобождение памяти

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 )
```

```
      { q->next = p->next;
```

```
        delete p;
```

```
    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) прочитать 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.

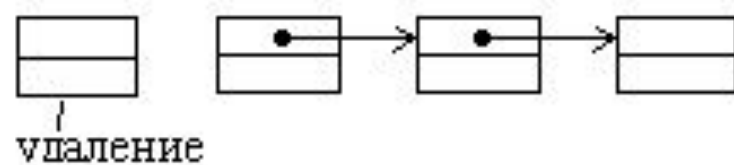
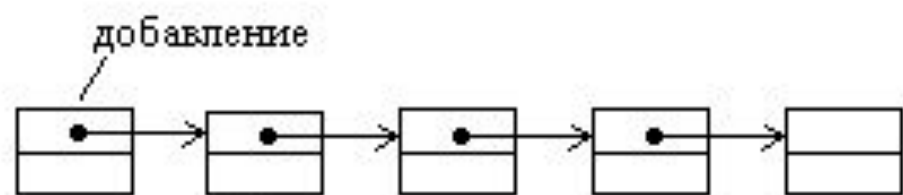
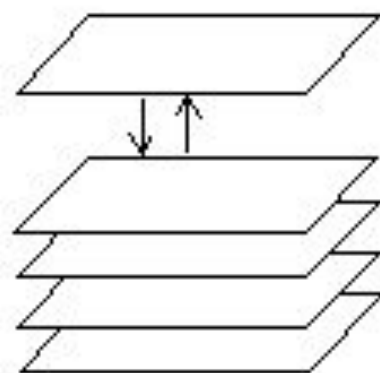
Что надо уметь делать со списком?

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

СТЕК

- **Стек** – это линейный список, в котором все включения и исключения (и обычно всякий доступ) делаются в одном конце списка (в голове списка).
- Стеки используются в работе алгоритмов, имеющих рекурсивный характер. Конец стека называется вершиной стека. Принцип работы стека - “последний пришел - первый вышел”. Внизу находится наименее доступный элемент. Часто говорят, что элемент опускается в стек.

Стек



Пример. Ввести с клавиатуры 10 чисел, записав их в стек. Вывести содержимое стека и очистить память.

```
#include <iostream>
using namespace std;
struct Node
{
    int x;//информационный элемент
    Node *Next;//указатель на следующий элемент
};
typedef Node *PNode;
```

```
void Add(int x,PNode &Head)//добавление элемента в стек
{
    PNode MyNode;
    if (Head==NULL)
    {
        Head=new(Node);
        MyNode=Head;
        Head->Next=NULL;
    }
    else
    {
        MyNode=new(Node);
        MyNode->Next=Head;
        Head=MyNode;
    }
    MyNode->x=x;
}
```

```
,
void Show(PNode &Head)//отображение стека
{
    PNode MyNode;
    MyNode=Head;//объявляем указатель на начало стека
    while (MyNode!=NULL)//пока указатель на следующий элемент не NULL
    {
        cout<<MyNode->x<<" ";//выводим поле
        MyNode=MyNode->Next;//переходим к следующему элементу
    }
}
void ClearNode(PNode &Head)//удаление стека из памяти
{
    PNode MyNode;
    while (Head!=NULL)//пока голова стека не указывает на NULL
    {
        MyNode=Head->Next;//переменная для хранения адреса следующего элемента
        delete Head;//освобождение адреса начала стека
        Head=MyNode;//меняем адрес начала стека
    }
}
```

```
void main()
```

```
{
```

```
    PNode Head;
```

```
    Head=NULL;
```

```
    for (int i=0;i<10;i++) //вносим данные в стек
```

```
        Add(i,Head);
```

```
    Show(Head); //выводим стек
```

```
    ClearNode(Head); //очищаем память
```

```
}
```

- **Очередь** – это линейный список, в один конец которого добавляются элементы, а с другого конца исключаются, элементы удаляются из начала списка, а добавляются в конце списка – как обыкновенная очередь в магазине. Принцип работы очереди: «первый пришел - первый вышел».

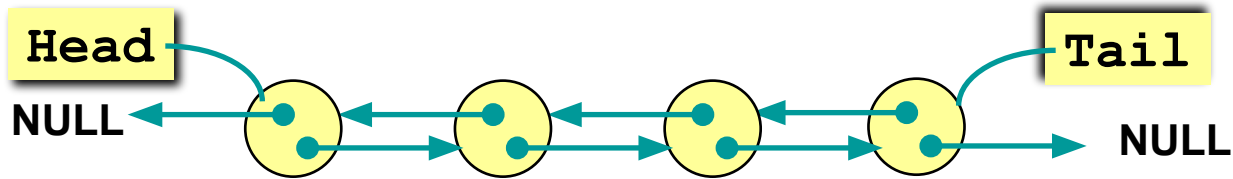
- Пример. Ввести с клавиатуры 10 чисел, записав их в очередь. Вывести содержимое очереди и очистить память.
- Для решения этой задачи достаточно в предыдущем примере изменить процедуру добавления элемента.

```
void Add(int x, PNode &Head, PNode &MyNode)//добавление элемента в очередь
{
    PNode Temp;
    if (Head==NULL)
    {
        Head=new(Node);
        MyNode=Head;
        Head->Next=NULL;
    }
    else
    {
        Temp=new(Node);
        MyNode->Next=Temp;
        MyNode=Temp;
        MyNode->Next=NULL;
    }
    cin>>MyNode->x;
}
```

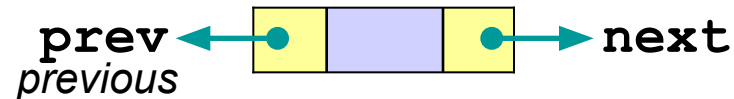
Основная программа:

```
void main()  
{  
    PNode Head, MyNode;  
    Head=NULL; MyNode=NULL;  
    for (int i=0;i<10;i++) //вносим данные в очередь  
        Add(i,Head,MyNode);  
    Show(Head); //выводим очередь  
    ClearNode(Head); //очищаем память  
}
```


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 «головы» и «хвоста»:

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



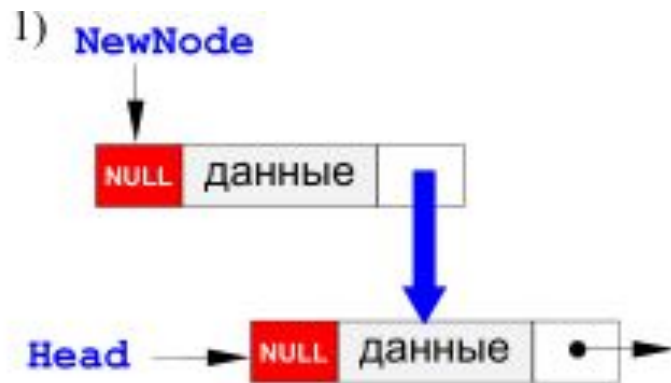
МОЖНО ДВИГАТЬСЯ В
обе стороны



нужно правильно
работать с двумя
указателями ВМЕСТО
одного

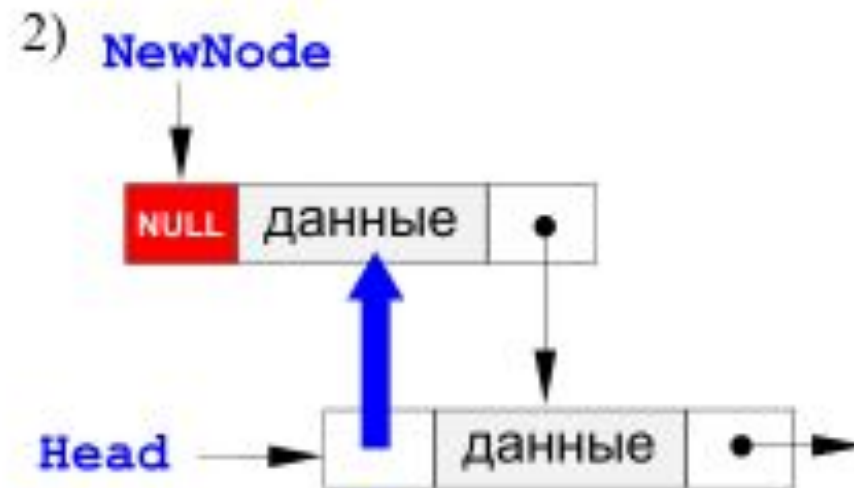
Двусвязные списки

- При добавлении нового узла `NewNode` в начало списка надо
- 1) установить ссылку `next` узла `NewNode` на голову существующего списка и его ссылку `prev` в `NULL`;



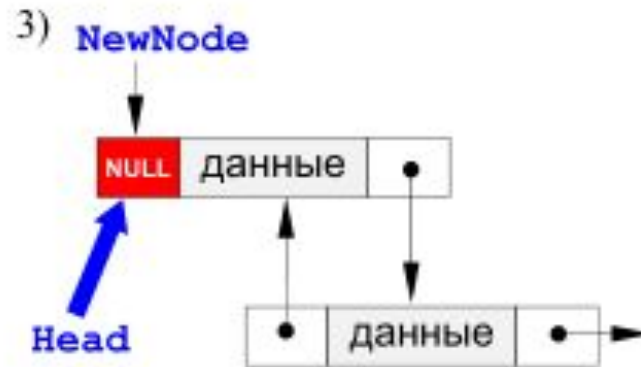
Двусвязные списки

2) установить ссылку prev бывшего первого узла (если он существовал) на NewNode;



Двусвязные списки

3) установить голову списка на **новый** узел;



4) если в списке не было ни одного элемента, хвост списка также устанавливается на **новый** элемент.

Двусвязные списки

```
void AddFirst(PNode &Head, PNode &Tail, PNode NewNode)
{
    NewNode->next = Head;
    NewNode->prev = NULL;
    if ( Head ) Head->prev = NewNode;
    Head = NewNode;
    if ( ! Tail ) Tail = Head; // этот элемент - первый
}
```

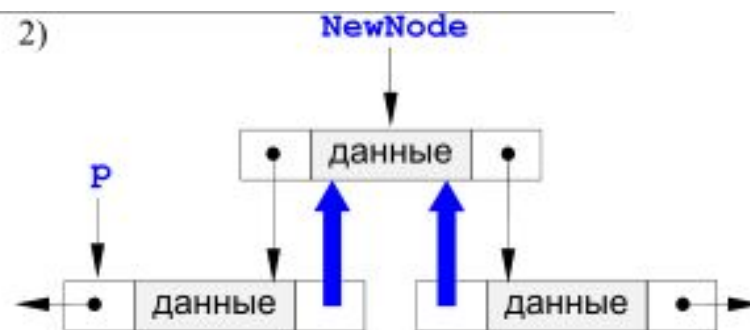
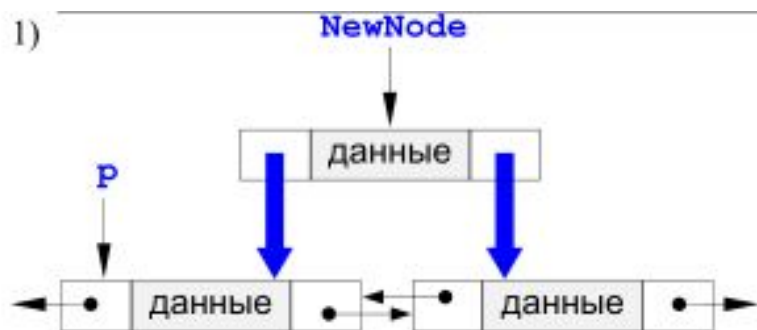
Добавление узла после заданного

- Дан адрес `NewNode` нового узла и адрес `p` одного из существующих узлов в списке.
- Требуется вставить в список новый узел **после `p`**.
- Если узел `p` является **последним**, то операция сводится к **добавлению в конец списка**.

Добавление узла после заданного

Если узел p – не последний, то операция вставки выполняется в два этапа:

- 1) установить ссылки нового узла на следующий за данным ($next$) и предшествующий ему ($prev$);
- 2) установить ссылки соседних узлов так, чтобы включить $NewNode$ в список.



Добавление узла после заданного (добавление после выполняется аналогично)

```
void AddAfter (PNode &Head, PNode &Tail,  
              PNode p, PNode NewNode)  
{  
    if ( ! p->next )  
        AddLast (Head, Tail, NewNode); // вставка в конец списка  
    else {  
        NewNode->next = p->next; // меняем ссылки нового узла  
        NewNode->prev = p;  
        p->next->prev = NewNode; // меняем ссылки соседних узлов  
        p->next = NewNode;  
    }  
}
```


Поиск узла в списке

Проход по двусвязному списку может выполняться в двух направлениях – от головы к хвосту (как для односвязного) или от хвоста к голове.

Проход по списку в от головы 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**
5. Перейти к п.2

```
...  
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  
}  
...
```

Проход по списку от хвоста к голове списка

Алгоритм:

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

```
PNode q = Tail;    // начали с хвоста  
while ( q != NULL )  
{           // пока не дошли до начала  
  ...      // делаем что-то хорошее с q  
  q = q->prev; // переходим к следующему узлу  
}  
...
```

Удаление узла



```
void Delete(PNode &Head, PNode &Tail, PNode OldNode)
{
    if (Head == OldNode) {
        Head = OldNode->next;    // удаляем первый элемент
        if ( Head )
            Head->prev = NULL;
        else Tail = NULL;        // удалили единственный элемент
    }
    else {
        OldNode->prev->next = OldNode->next;
        if ( OldNode->next )
            OldNode->next->prev = OldNode->prev;
        else Tail = OldNode->prev;    // удалили последний элемент
    }
    delete OldNode;
}
```


Циклические списки

- Иногда список (односвязный или двусвязный) замыкают в кольцо.
- Указатель `next` последнего элемента указывает на первый элемент.
- Для двусвязных списков указатель `prev` первого элемента указывает на последний. В таких списках понятие «хвоста» списка не имеет смысла, для работы с ним надо использовать указатель на «голову», причем «головой» можно считать любой элемент.

- Дано число N и M целых чисел. Создать циклический односвязный линейный список, в который поместить все эти элементы в порядке поступления (тип – очередь). Вернуть указатель на первый элемент списка. Затем распечатать этот список.

```
void create_cycle (PNode &head, int n)
{
    PNode p,q;
    head=new Node;
    cout<<"Значение ";
    cin>>head->inf;
    p=head;
    head->next=p;
    for(int i=2;i<=n;i++)
    {
        q=new Node;
        cout<<"Значение ";
        cin>>q->inf;
        p->next=q;
        q->next=head;
        p=q;
    }
}
```

```
void print_cycle (PNode head)
{
    PNode p;
    cout<<head->inf<<'\\t';
    p=head->next;
    while (p !=head)           // пока не голова списка
    {
        cout<<p->inf<<'\\t';
        p=p->next;    // переходим к следующему узлу
    }
    cout<<endl;
}
```

Вывести значение N -го элемента кольцевого списка. Считать, что счет начинается с первого элемента и продолжается по кругу, если элементов в списке меньше N .

```
void browse_loop_N (PNode head, int N)
{
    for (int i=2; i<=N; i++)
        head=head->next;
    cout<<head->inf<<endl;
}
```

Вставить значение x после N -го по счету элемента циклического списка.

```
void insert_loop_N(PNode head, int N, int x)
{
    for (int i=1;i<=N-1;i++)
        head=head->next;
    PNode p;
    p=new Node;
    p->inf=x;
    p->next=head->next;
    head->next=p;
}
```