ФГБОУ ВО РГУПС

Алгоритмизация и программирование

Динамические структуры данных. Двухсвязные списки

Лекция 9

Ростов-на-Дону
2020

План лекции

- Двухсвязный список
- Циклический список

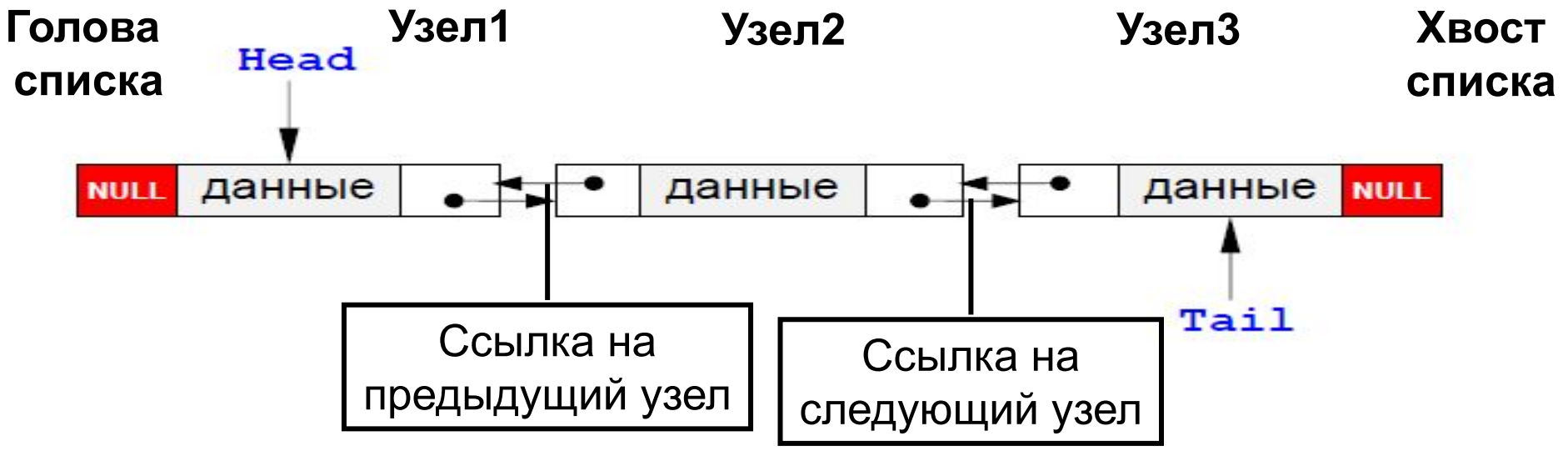
Двухсвязные списки

Двухсвязные списки

- **Двухсвязный список – это динамический список, в котором каждый узел содержит две ссылки**.
- Каждый элемент содержит **ссылку на следующий и предыдущий** элемент.
- Вводятся две переменные-указатели – ссылка на «голову» списка (**Head**) и на «хвост» - последний элемент (**Tail**).

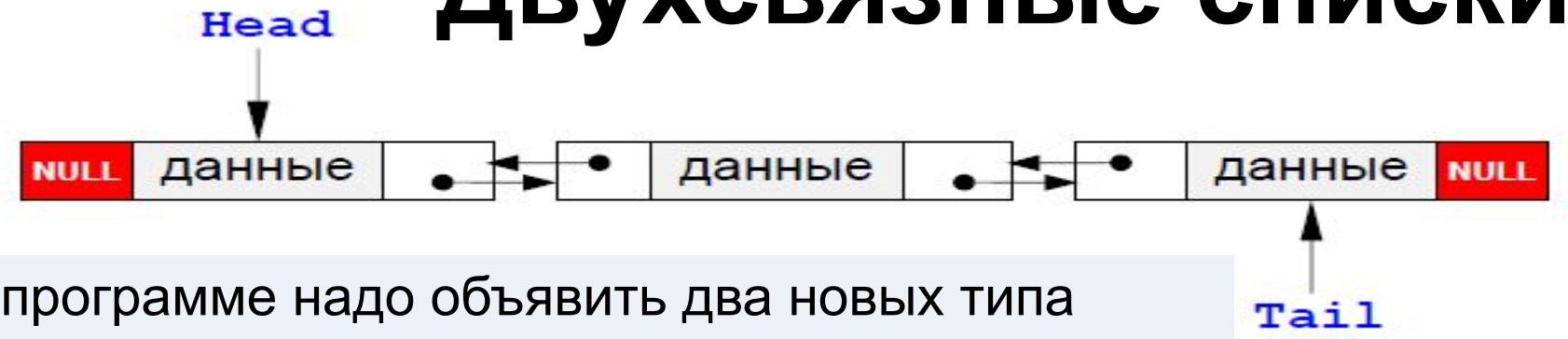
Двухсвязные списки

Схема двухсвязного списка



Узел представляет собой структуру, которая содержит поля данные и 2 указателя на следующий и предыдущий узел.

Двухсвязные списки



В программе надо объявить два новых типа данных – узел списка **Node** и указатель на него **PNode**.

Синтаксис объявления двухсвязного списка в C++:

```
struct Node
{
    Тип1  поле1;
    Тип2  поле2;
     …..
     Node *next;
     Node *prev;
};
Node * PNode;
```

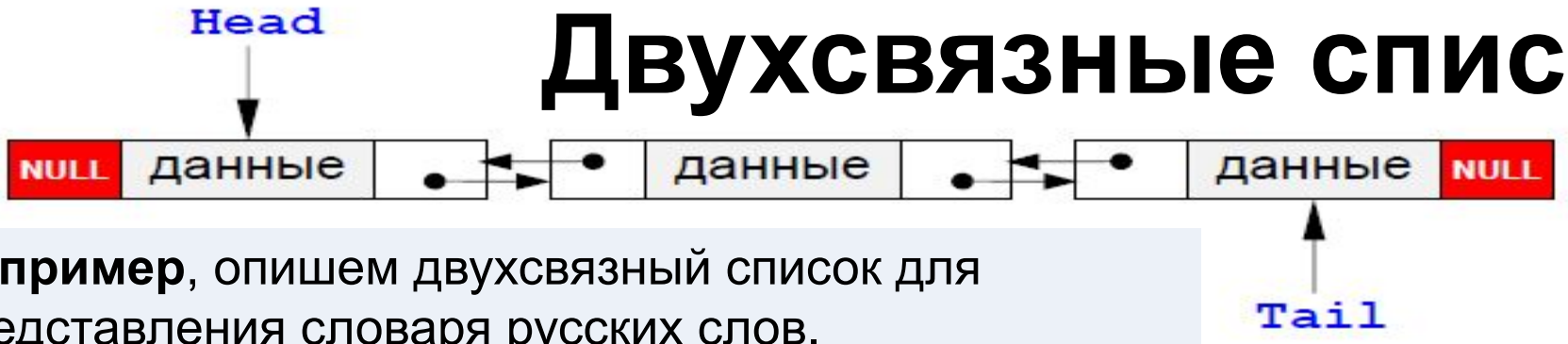
область данных

ссылка на следующий узел

ссылка на предыдущий узел

указатель на узел

Двухсвязные списки



Например, опишем двухсвязный список для представления словаря русских слов.
Узел представляет собой структуру, которая содержит три поля – строку и два указателя на следующий и предыдущий узел.

Пример. Объявление двухсвязного списка словаря:

```
struct Node
{
    string  word;
    Node *next;
    Node *prev;
};
typedef Node *PNode;
PNode Head = NULL;
PNode Tail = NULL;
```

Область данных

ссылка на следующий узел

ссылка на предыдущий узел

тип данных - указатель на узел

Указатель на начало списка. В начале работы указатель Head равен NULL

Указатель на конец списка. В начале работы указатель Tail равен NULL

Двухсвязные списки. Операции

Операции над двухсвязными списками:

- Создание нового узла.
- Добавление узла:
  - В начало списка
  - В конец списка
  - После заданного узла
  - Перед заданным узлом
- Проход по списку
- Поиск узла
- Удаление узла

Создание нового узла

- Для того, чтобы добавить узел к списку, необходимо **создать** его, то есть **выделить память под узел и запомнить адрес выделенного блока.**
- Будем считать, что надо добавить к списку узел, соответствующий новому слову, которое записано в переменной **NewWord.**
- Составим **функцию**, которая создает новый узел в памяти и возвращает его адрес. При записи данных в узел используется обращение к полям структуры через указатель.

А что нужно изменить для двухсвязного списка?

```
Pnode  CreateNode ( string  NewWord )
{       PNode NewNode = new Node;
        NewNode->word = NewWord;
        NewNode->next = NULL;
        NewNode->prev = NULL;
        return NewNode;
}
```

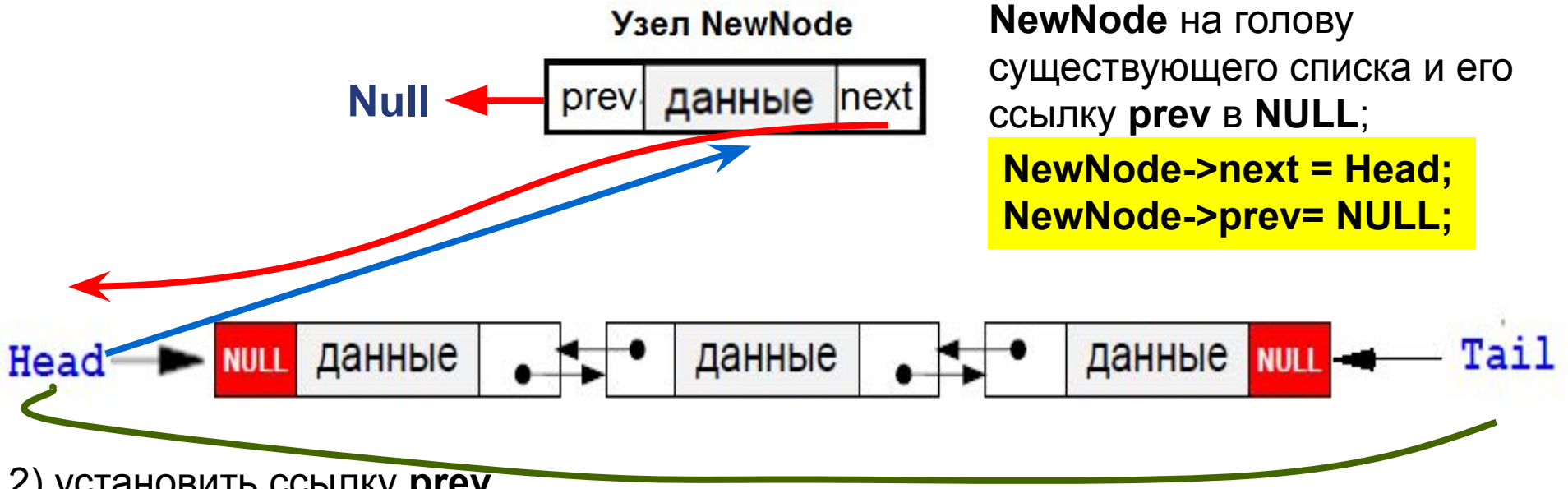
```
// функция создания узла
// указатель на новый узел
// записать слово

// следующего узла нет
// предыдущего узла нет

// результат функции – адрес узла
```

Добавление узла в начало списка

- 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**;

```
NewNode->next = Head;
NewNode->prev= NULL;
```

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

```
if ( Head!=NULL )
   Head->prev = NewNode;
```

3) установить голову списка на новый узел;

```
Head = NewNode;
```

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

```
if ( Tail == NULL ) Tail = Head;
```

Добавление узла в начало списка

- По такой схеме работает функция **AddFirst**.
- Важно, что здесь и далее **адрес начала и конца списка передаются *по ссылке***, так как при добавлении нового узла они изменяются внутри функции.

// функция добавления нового узла в начало списка

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

1) установить ссылку next узла NewNode на голову существующего списка и его ссылку prev в NULL;

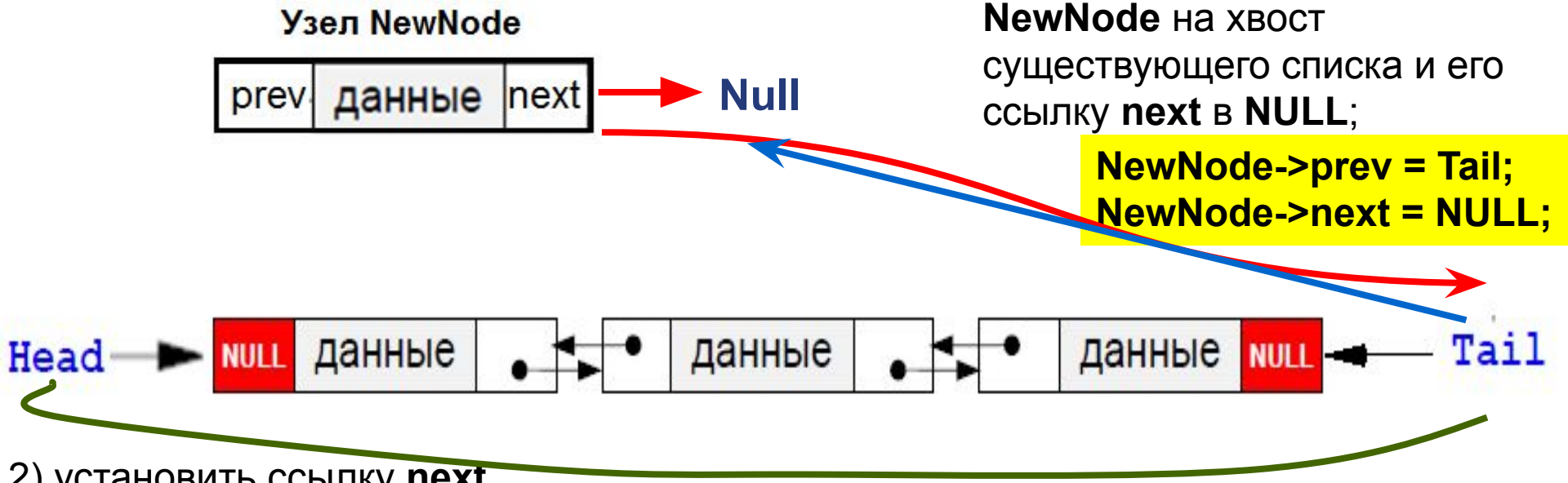
2) установить ссылку prev бывшего первого узла (если он существовал) на NewNode;

3) установить голову списка на новый узел;

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

Добавление узла в конец списка

- Благодаря симметрии добавление нового узла **NewNode** в конец списка проходит совершенно аналогично, в процедуре надо везде заменить **Head** на **Tail** и наоборот, а также поменять **prev** и **next**.



1) установить ссылку **prev** узла **NewNode** на хвост существующего списка и его ссылку **next** в **NULL**;

```
NewNode->prev = Tail;
NewNode->next = NULL;
```

2) установить ссылку **next** бывшего конечного узла (если он существовал) на **NewNode**;

```
if ( Tail!=NULL )
  Tail->next = NewNode;
```

3) установить хвост списка на новый узел;

```
Tail = NewNode;
```

4) если в списке не было ни одного элемента, голову списка также устанавливается на новый узел.

```
if ( Head == NULL )
    Head = Tail;
```

Добавление узла в конец списка

- По такой схеме работает функция **AddLast**.

// функция добавления нового узла в конец списка

Была функция AddFirst

Стала функция AddLast

```
void AddLast (PNode &Head, PNode &Tail, PNode NewNode)
{
  NewNode->next = Head;
  NewNode->prev = NULL;
  if ( Head != NULL )
    Head->prev = NewNode;
  Head = NewNode;
  if ( Tail == NULL ) Tail = Head;
}
```

```
NewNode->prev = Tail;
NewNode->next = NULL;
if ( Tail != NULL )
  Tail->next = NewNode;
Tail = NewNode;
 if (Head == NULL) Head = Tail;
```

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 узла с адресом **p**.

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***);

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

Если узел **p** является последним, то операция сводится к добавлению в конец списка

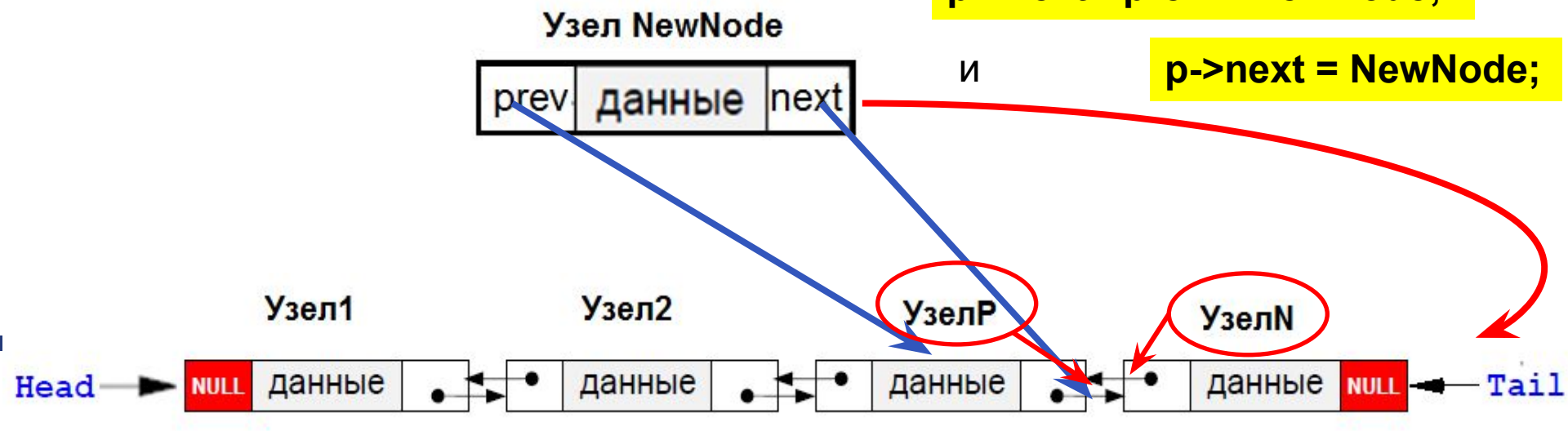
```
if ( p->next  == NULL)
AddLast (Head, Tail, NewNode);
```

2) установить ссылки соседних узлов так, чтобы включить **NewNode** в список.

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

и

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



Добавление узла после заданного

- По такой схеме работает функция **AddAfter**.
- Передавать будем адрес узла после которого, хотим вставить новый узел и адрес самого нового узла.

// функция добавления нового узла после указанного

```
void AddAfter (PNode &Head, PNode &Tail, PNode p, PNode
NewNode)
{    if ( p->next == NULL)
        AddLast (Head, Tail, NewNode);
   else
     {
        NewNode->next = p->next;
        NewNode->prev = p;
        p->next->prev = NewNode;
        p->next = NewNode;
     }
}
```

1) Если узел **p** является последним, то вставить **NewNode** в конец списка.

2) Иначе:
а) установить ссылки нового узла на следующий за данным (***next***) и предшествующий ему (***prev***);
б) установить ссылки соседних узлов так, чтобы включить **NewNode** в список.

Добавление узла перед заданным

- Добавление узла перед заданным выполняется аналогично, как и после заданного, и является симметричной операцией!!

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***);

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

Если узел **p** является первым, то операция сводится к добавлению в начало списка

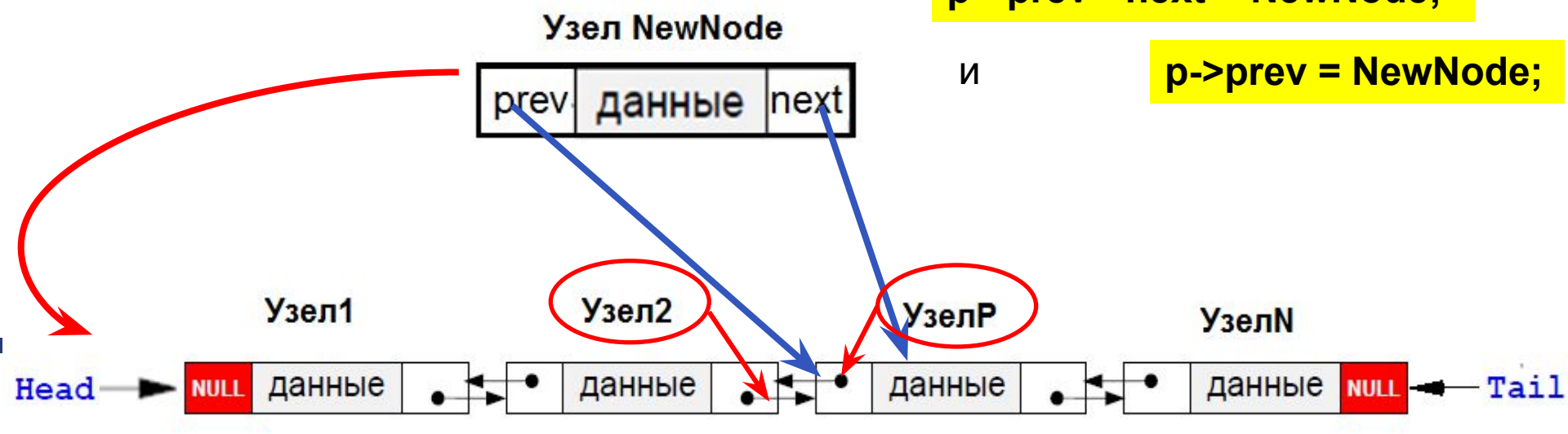
```
if ( p  == Head)
AddFirst (Head, Tail, NewNode);
```

2) установить ссылки соседних узлов так, чтобы включить **NewNode** в список.

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

и

```
p->prev = NewNode;
```



Добавление узла перед заданного

- По такой схеме работает функция **AddBefore**.
- Передавать будем адрес узла перед которого, хотим вставить новый узел и адрес самого нового узла.

// функция добавления нового узла после указанного

```
void AddBefore (PNode &Head, PNode &Tail, PNode p, PNode
NewNode)
{    if ( p == Head)
       AddFirst (Head, Tail, NewNode);
   else
    {
       NewNode->next = p;
       NewNode->prev = p->prev;
       p->prev->next = NewNode;
       p->prev = NewNode;
    }
}
```

1) Если узел **p** является первым, то вставить **NewNode** в начало списка.

2) Иначе:
а) установить ссылки нового узла на следующий за данным (***next***) и предшествующий ему (***prev***);
б) установить ссылки соседних узлов так, чтобы включить **NewNode** в список.

Проход по списку

Проход по двусвязному списку может выполняться в **двух направлениях :**

– **от головы к хвосту** (как для односвязного) , используя указатель **next**, продвигаться к следующему узлу.

- **от хвоста к голове**, используя указатель **prev,** продвигаться к предыдущему узлу.

// Обход списка с головы списка

```
PNode p = Head;

while ( p != NULL )
{
    p = p->next;
}
```

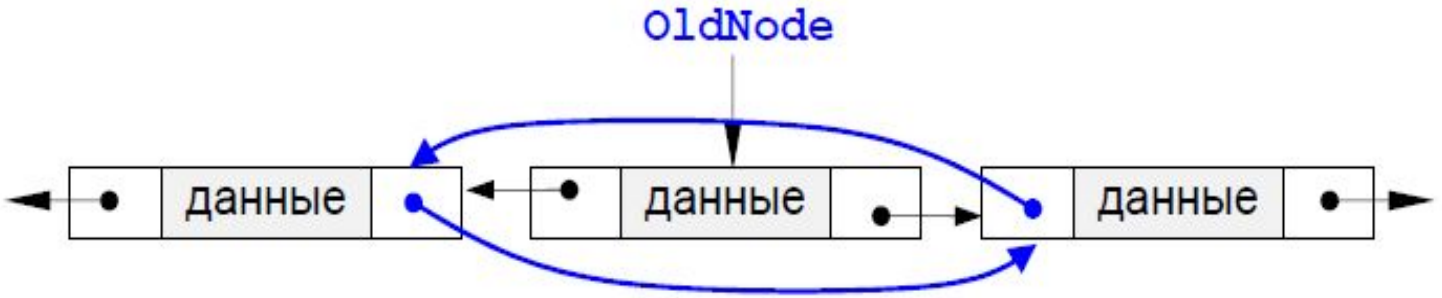
// Обход списка с хвоста списка

```
// PNode p = Tail;

//
// while ( p != NULL )
{
//     p = p->prev;
}
```

Удаление узла

- Эта процедура также требует ссылки на голову и хвост списка, поскольку они могут измениться при удалении крайнего элемента списка.
- На первом этапе устанавливаются ссылки соседних узлов (если они есть) так, как если бы удаляемого узла не было бы. В отличие от односвязного списка не нужно искать элемент предыдущий за удаляемым, так как у него есть ссылка **prev**!
- Затем узел удаляется и память, которую он занимает, освобождается.



Удаление узла

1) Рассмотрим крайний случай – если удаляемый элемент OldNode является первым!!

```
if (Head == OldNode)
{
    Head = OldNode->next;
```

2) Проверяем , это был единственный элемент или нет. Если голова не пустая, то устанавливаем предыдущую ссылку головы, а иначе все пусто!

```
 if ( Head  != NULL )
        Head->prev = NULL;
    else Tail = NULL;
}
```

3) Иначе, это не первый элемент. Изменяем ссылки соседних узлов

```
OldNode->prev->next = OldNode->next;
```

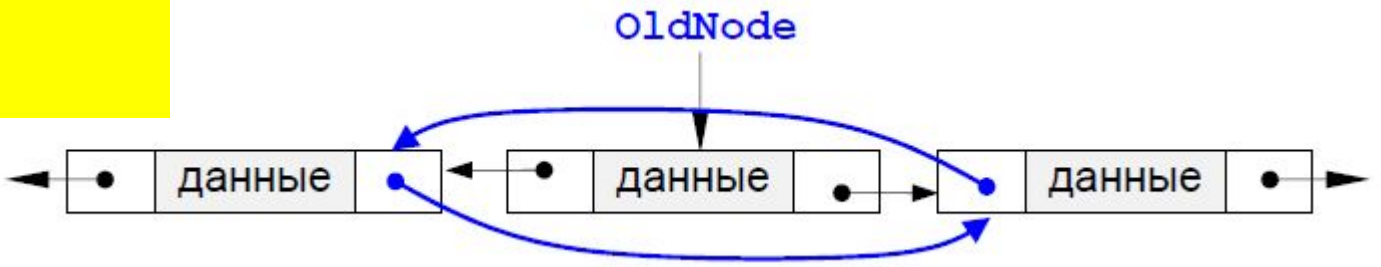
```
if ( OldNode->next !=NULL)
  OldNode->next->prev = OldNode->prev;
```

4) Иначе, удалили последний элемент

```
Tail = NULL;
```

5) Освобождаем память, которую занимал узел.

```
delete OldNode;
```



Удаление узла

```
void DeleteNode(PNode &Head, PNode &Tail, PNode OldNode)
{       if (Head == OldNode)          // удаляем первый элемент
        {    Head = OldNode->next;
    if ( Head !=NULL)
         Head->prev = NULL;
    else
         Tail = NULL;              // удаляем единственный элемент
    }
    else          // изменяем ссылки соседних элементов
    {   OldNode->prev->next = OldNode->next;
        if ( OldNode->next !=NULL)
         OldNode->next->prev = OldNode->prev;
        else
         Tail = NULL;                // удаляем последний элемент
    }
    delete OldNode;              // освобождаем память
}
```

Поиск узла в списке

Часто требуется найти в списке нужный элемент (его адрес или данные). Надо учесть, что требуемого элемента может и не быть, тогда просмотр заканчивается при достижении конца списка.
Такой подход приводит к следующему **алгоритму**:

1) начать с головы списка;

```
PNode q = Head;
```

2) пока текущий элемент существует (указатель – не NULL), проверить нужное условие и перейти к следующему элементу;

```
while (q->word != NewWord  && q!=NULL)
        q = q->next;
```

3) закончить, когда найден требуемый элемент или все элементы списка просмотрены.

```
return q;
```

Поиск по данным

Поиск узла в списке

Например, следующая функция ищет в списке элемент, соответствующий заданному слову (для которого поле **word** совпадает с заданной строкой **NewWord**), и возвращает его адрес или **NULL**, если такого узла нет.

```
// функция поиска узла в списке
PNode Find (PNode Head, string NewWord)
{    //начать с головы списка
     PNode q = Head;

     while (q->word != NewWord && q != NULL)
          q = q->next;

     return q;
}
```

//пока текущий элемент существует (указатель – не NULL), проверить нужное условие и перейти к следующему элементу

закончить, когда найден требуемый элемент или все элементы списка просмотрены

Поиск узла по порядку в списке

- Вернемся к задаче построения алфавитного словаря. Для того, чтобы добавить новое слово в нужное место (**в алфавитном порядке**), требуется **найти адрес узла, *перед* которым надо вставить новое слово**.
- Это будет первый от начала списка узел, для которого «его»
- слово окажется «больше», чем новое слово.
- Поэтому достаточно просто изменить условие в цикле **while** в функции **Find**, учитывая, что сравнение **q->word < NewWord** возвращает значение «больше» или «меньше» по естественному лексикографичексому порядку.

Поиск по данным в определённом порядке - алфавитном

Поиск узла по порядку в списке

1) начать с головы списка;

```
PNode q = Head;
```

2) пока текущий элемент существует (указатель – не NULL), проверить условие и перейти к следующему элементу;

```
while (q->word  <  NewWord  && q!=NULL)
      q = q->next;
```

3) закончить, когда найден требуемый элемент или все элементы списка просмотрены.

```
return q;
```

Куда его вставить?

Будем искать место – адрес узла, перед которым нужно вставить новый узел.
Чтобы его данные NewWord были меньше, чем данные у узла перед ним.

Поиск узла по порядку в списке

Эта функция вернет адрес узла, перед которым надо вставить новое слово , когда сравнение вернет true, или **NULL**, если слово надо добавить в конец списка.

// функция поиска узла по порядку в списке

```
PNode FindPlace (PNode Head, string NewWord)
{    //начать с головы списка
     PNode q = Head;

     while (q->word < NewWord && q != NULL)
          q = q->next;

     return q;
}
```

//пока текущий элемент существует (указатель – не NULL), проверить нужное условие и перейти к следующему элементу

закончить, когда найден требуемый элемент или все элементы списка просмотрены

Пример программы на двухсвязный линейный список

Пример. Двухсвязный список словарь

```
#include <iostream>
#include <string>
using namespace std;
// описание динамической структуры
struct Node
{
     string word;
     int count;
     Node *next;
     Node *prev;
};
typedef Node *PNode;
// Создание элемента списка
PNode CreateNode ( string NewWord )
{
   PNode NewNode = new Node;
   NewNode->word= NewWord;
   NewNode->count = 1;
   NewNode->next = NULL;
   NewNode->prev= NULL;
   return NewNode; /
}
 // и так далее описание всех функций
```

Пример (продолжение)

```
int main()
{
    PNode Head = NULL, Tail = NULL;
    PNode pnew, pfind;
    int t; string newslovo;
    do
    {
      cout<<"введите от 1 до 5 или 0 - выход"<<endl;
      cout<<" 0 - выход "<<endl;
      cout<<" 1 - добавить новый элемент в конец списка "<<endl;
      cout<<" 2 - вывод списка "<<endl;
      cout<<" 3 - добавить новый элемент после выбранного "<<endl;
      cout<<" 4 - добавить новый элемент по алфавиту "<<endl;
      cout<<" 5 - добавить новый элемент перед выбранным "<<endl;
      cout<<" 6 - удалить элемент  "<<endl;
      cout<<endl;
            cin>>t;
            switch (t)
      {

}
```

Пример (продолжение)

```
case 1 :
                    cout<<"введите новое слово = ";
          cin>>newslovo;
          pnew=CreateNode(newslovo); // создаем новый узел
          if (Head==NULL)
               AddFirst (Head, Tail, pnew) ;    //вставляем на первое место
          else
               AddLast (Head, Tail, pnew) ;    //вставляем в конец списка
          break;
     case 2 :
          pnew=Head;  // вывод списка на экран
          while (pnew!=NULL)
           {
                cout<<pnew->word<<"\t"<<pnew->count<<endl;
                pnew=pnew->next;
           }
          break;
        case 3:
          /////
          };
     }
  while (t!=0);
  return 0;
```

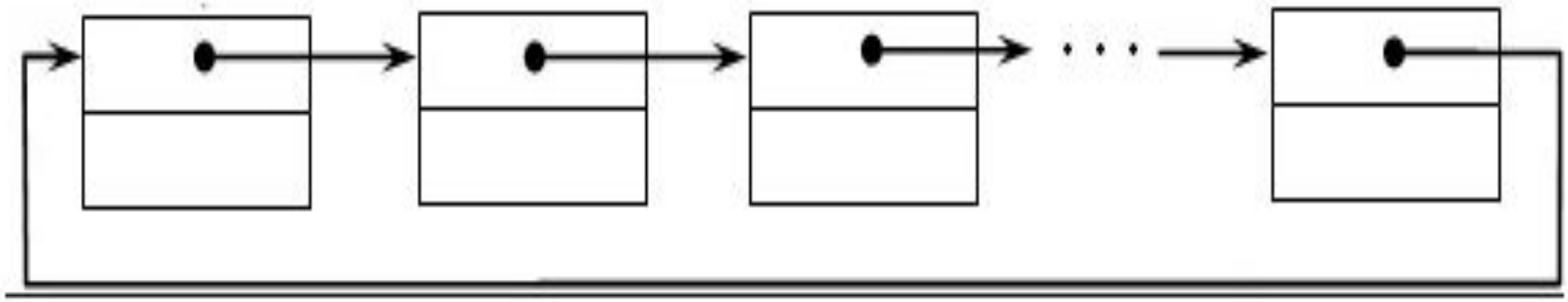
Циклические списки

Циклические списки

- Иногда список (односвязный или двусвязный) замыкают в кольцо, то есть указатель **next** последнего элемента указывает на первый элемент, и (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.

Циклический список

- **Замкнутый (кольцевой, циклический) список** — головной и хвостовой элементы которого указывают друг на друга.



Спасибо за внимание!

Литературные источники

- 1) К. Поляков. Программирование на языке C++.
- 2). Харви Дейтел, Пол Дейтел. Как программировать на C++. - М: Вильямс, - 1011 с.
- 3). Структруп Б. Программирование: принципы и практика использования C++. – М. : Вильямс, 2011. – 1248 с.
- 4). Структруп Б. Язык программирования C++. – М.: Бином. - 1054 с.
- 5). Лафоре Р. Объектно-ориентированное программирование в C++. Питер, 2004. – 922 с.
- 6). Шилдт Г. C++: руководство для начинающих, 2-е издание. – М: Вильямс, 2005. -672 с.